AFRL-IF-RS-TR-2002-207
Final Technical Report
August 2002

SA-CIRCA: SELF-ADAPTIVE CONTROL FOR MISSION-CRITICAL SYSTEMS

Honeywell Technology Center

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. G428

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-2002-207 has been reviewed and is approved for publication

APPROVED: [illegible]

JOHN J. CROWTER
Project Engineer

FOR THE DIRECTOR: James A. Collins

JAMES A COLLINS
Acting Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** Aug 02	**3. REPORT TYPE AND DATES COVERED** Final Jun 98 – Nov 99

4. TITLE AND SUBTITLE SA-CIRCA: SELF-ADAPTIVE CONTROL FOR MISSION-CRITICAL SYSTEMS	**5. FUNDING NUMBERS** C - F30602-98-C-0212 PE - 62301E/62702F/63728F PR - G428 TA - 01 WU - 01
6. AUTHOR(S) David J. Musliner, Robert P. Goldman, Michael J. Pelican, Kurt D. Krebsbach, Edmund H. Dunfee, Kang G. Shiu, Haksun Li and Ella Atkins	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Honeywell Technology Center 3660 Technology Drive Minneapolis, MN 55418	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency — AFRL/IFTD 3701 North Fairfax Drive — 525 Brooks Rd Arlington, VA 22203-1714 — Rome, NY 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2002-207

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: John J. Crowter, IFTB, 315-330-1459, crowterj@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

The goal of this effort was to begin extending the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Self-Adaptive CIRCA (SA-CIRCA). CIRCA is a coarse-grain architecture designed to control autonomous systems which require both intelligence, deliberative planning activity and highly reliable, hard-real-time reactions to safety threats. CIRCA allows systems to provide performance guarantees that ensure they will remain safe and accomplish mission-critical goals while also intelligently pursuing long-term, non-critical goals. The SA-CIRCA project took several steps towards extending this architecture with the ability to reason accurately about its own real-time behavior, and adapt that behavior in response to performance feedback. Due to a change in the direction of this research, the SA-CIRCA project was only partially funded. As a result, the development of the architecture and demonstrations was not completed. Major issues investigated during this project include formally verifying real-time control plans, dynamically decomposing long-term plans into sub-goals, and building real-time control plans using probabilistic information to reason about most-likely states first. The primary technical products of this research project are two versions of CIRCA's controller-synthesis (or planning) algorithm. The first version automatically generates reactive control plans and verifies their correctness using formal model-checking methods. The second version does not use model checking to verify its plans, but uses a novel form of probabilistic reasoning to restrict its planning effort to the most-likely future system state.

14. SUBJECT TERMS Mission-Critical Systems, Self-Adaptive Systems, unmanned Autonomous Vehicles, Mission Planning, Machine Learning, Artificial Intelligence	**15. NUMBER OF PAGES** 48
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

PREFACE

The work reported here was conducted by the Honeywell Technology Center, Minneapolis, MN, during the period June, 1998 through December 1999 under Rome Air Force Research Laboratory contract F30602-98-C-0212. Mr. John J. Crowter was the AFRL project officer for the contract.

SA-CIRCA: Self-Adaptive Control for Mission-Critical Systems

SUMMARY

This is the final report for the Defense Advanced Research Projects Agency (DARPA) contract F30602-98-C-0212 entitled "SA-CIRCA: Self-Adaptive Control for Mission-Critical Systems." The goal of this contract effort was to begin extending the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Self-Adaptive CIRCA (SA-CIRCA). CIRCA is a coarse-grain architecture designed to control autonomous systems which require both intelligent, deliberative planning activity and highly reliable, hard-real-time reactions to safety threats. CIRCA allows systems to provide performance guarantees that ensure they will remain safe and accomplish mission-critical goals while also intelligently pursuing long-term, non-critical goals. The SA-CIRCA project took several steps towards extending this architecture with the ability to reason accurately about its own real-time behavior, and adapt that behavior in response to performance feedback. The SA-CIRCA project was only partially funded, so the development of the architecture and demonstrations was not completed. Major issues investigated during this project include formally verifying real-time control plans, dynamically decomposing long-term plans into subgoals, and building real-time control plans using probabilistic information to reason about most-likely states first.

The primary technical products of this research project are two versions of CIRCA's controller-synthesis (or planning) algorithm. The first version, developed by Honeywell, automatically generates reactive control plans and verifies their correctness using formal model-checking methods. The second version, developed by the University of Michigan on subcontract, does not use model checking to verify its plans, but uses a novel form of probabilistic reasoning to restrict its planning effort to the most-likely future system states.

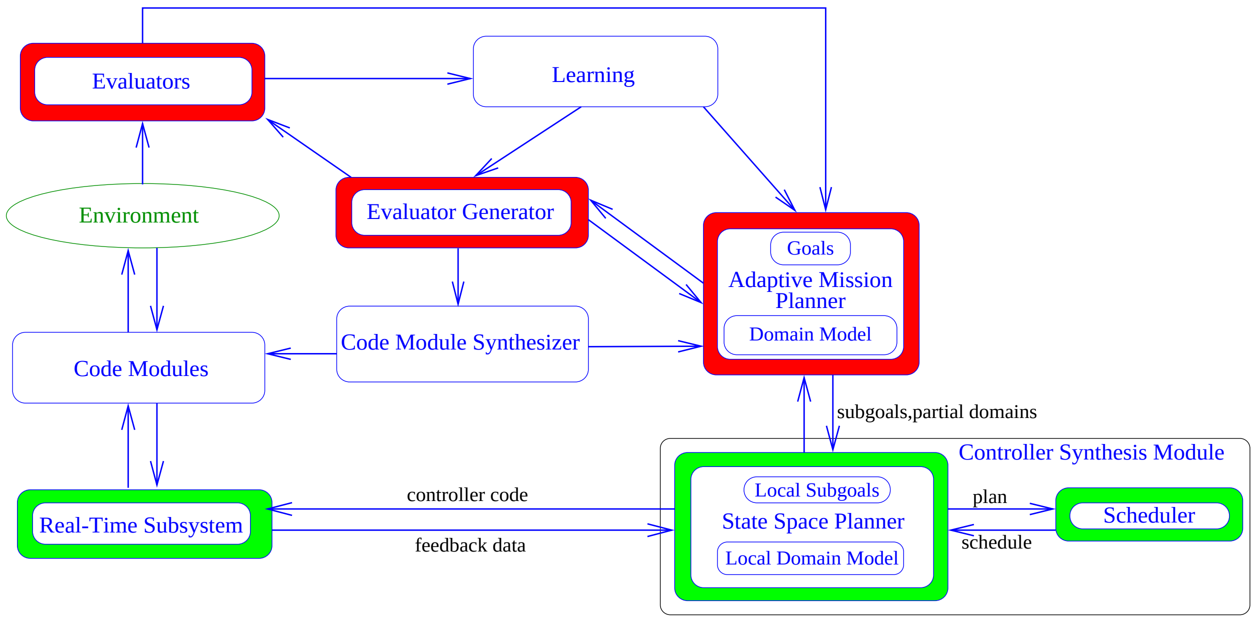


Figure 1. The SA-CIRCA architecture.

1. Introduction

This is the final report for the Defense Advanced Research Projects Agency (DARPA) contract F30602-98-C-0212 entitled "SA-CIRCA: Self-Adaptive Control for Mission-Critical Systems." The goal of this contract effort was to begin extending the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Self-Adaptive CIRCA (SA-CIRCA). CIRCA is a coarse-grain architecture designed to control autonomous systems which require both intelligent, deliberative planning activity and highly reliable, hard-real-time reactions to safety threats. CIRCA allows systems to provide performance guarantees that ensure they will remain safe and accomplish mission-critical goals while also intelligently pursuing long-term, non-critical goals. The SA-CIRCA project took several steps towards extending this architecture with the ability to reason accurately about its own real-time behavior, and adapt that behavior in response to performance feedback.

1.1. The SA-CIRCA Architecture

As illustrated in Figure 1, SA-CIRCA was designed to meet the requirements for real-time self-adaptive control software operating autonomously and safely for extended periods of time in dynamic environments. Building on the proven real-time planning and control capabilities of the original CIRCA architecture (lightly shaded in Figure 1), SA-CIRCA was designed to provide revolutionary self-modeling, performance evaluation, and adaptation functions through several key new technology developments (dark shading):

- An **Adaptive Mission Planner** that manages long-term mission planning and adaptation, developing mission plans while reasoning about performance evaluation, evaluator feedback, resource restrictions, and dynamic goals.

- An **Evaluator Generator** that analyzes plans and monitors system goals, and builds sentinel processes that monitor system activities and the environment to recognize threats to goal achievement and opportunities to optimize performance.
- **Performance Evaluators** that provide descriptions of system failures, or operations/capabilities that would improve system performance. These descriptions can be used to pull existing code from a library, or as inputs to a Code Module Synthesizer.

These new capabilities were designed to add self-evaluation and long-term self-adaptive behavior to the existing CIRCA architecture, which provides:

- A **Real-Time Subsystem** (RTS) that predictably executes real-time control plans.
- A **State Space Planner** and **Scheduler** that cooperate to reason about internal models of the world and dynamically program the RTS with a planned set of reactions guaranteed to keep the system safe. These control plans are called TAP (Test Action Pair) schedules. Together, the SSP and Scheduler are components of the Controller Synthesis Module (CSM).

1.2. Progress

The SA-CIRCA project was only partially funded, so the development of all the planned architecture features and demonstrations was not completed. Major issues investigated during this project include formally verifying real-time control plans, using probabilistic reasoning to control the complexity of control plan synthesis, and dynamically decomposing long-term plans into subgoals.

This research project has yielded two primary technical products:

- An integrated controller-synthesis (or planning) algorithm that automatically generates reactive control plans and verifies their correctness using formal model-checking methods (see Section 3).
- An alternative controller-synthesis (or planning) algorithm that uses probabilistic information to plan for the most-probable contingencies first (see Section 4).

In addition, we conducted preliminary investigations into the design of SA-CIRCA's top level component, the Adaptive Mission Planner, using two existing AI planning systems (see Section 5). Section 6 describes the simulated UAV demonstrations we developed to illustrate the behavior of SA-CIRCA controllers in real-time, mission-critical environments.

2. Overview of CIRCA

CIRCA is designed to support both hard real-time response guarantees and unrestricted Artificial Intelligence (AI) methods that can guide those real-time responses. Figure 1

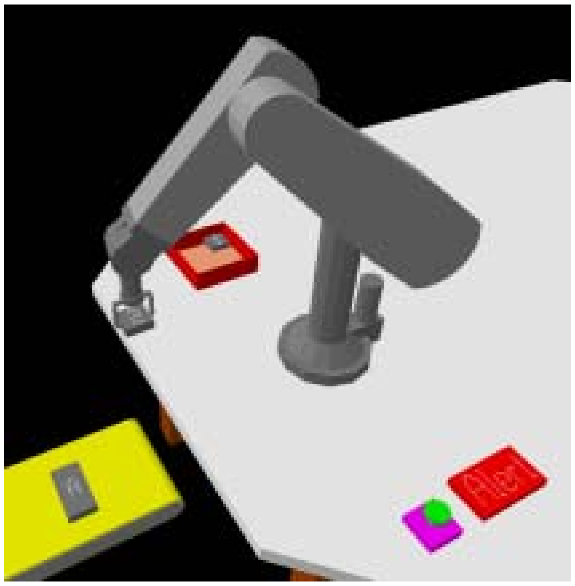


Figure 2. The simulated Puma robot arm domain.

illustrates the architecture, in which the AMP and CSM reason about high-level problems that require their powerful but potentially unbounded planning methods, while a separate Real-Time Subsystem (RTS) reactively executes the automatically-generated plans and enforces guaranteed response times. The AMP and CSM modules cooperate to develop executable reaction plans that will assure system safety and attempt to achieve system goals when interpreted by the RTS.

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics and simulated unmanned aircraft (UAVs). A UAV example will be discussed in detail in Section 3. To introduce the key CIRCA concepts in this section, we draw examples from the domain illustrated by Figure 2, in which CIRCA controls a simulated Puma robot arm that must pack parts arriving on a conveyor belt into a nearby box. The parts can have several shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not initially know how to pack all of the possible types of parts— it may have to perform some computation to derive an appropriate box-packing strategy. The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system must push the button next to the light before a fixed deadline.

In this domain, CIRCA's planning and execution subsystems operate in parallel. The CSM reasons about an internal model of the world and dynamically programs the RTS with a planned set of reactions. While the RTS is executing those reactions, ensuring that the system avoids failure, the AMP and CSM are able to continue executing heuristic planning methods to find the next appropriate set of reactions. For example, the AMP may derive a new box-packing algorithm that can handle a new type of arriving part. The derivation of this new algorithm does not need to meet a hard deadline, because the reactions concurrently executing on the RTS will continue handling all arriving parts, just stacking

```
EVENT emergency-alert                          ;; Emergency light goes on
      PRECONDS: ((emergency nil))
      POSTCONDS: ((emergency T))

TEMPORAL emergency-failure                     ;; Fail if don't attend to
      PRECONDS: ((emergency T))                ;; light by deadline
      POSTCONDS: ((failure T))
      MIN-DELAY: 30 [seconds]

ACTION push-emergency-button
      PRECONDS: ((part-in-gripper nil))
      POSTCONDS: ((emergency nil) (robot-position over-button))
      WORST-CASE-EXEC-TIME: 2.0 [seconds]
```

Figure 3. Example transition descriptions given to CIRCA's planner.

unfamiliar ones on a nearby table temporarily. When the new box-packing algorithm has been developed and integrated with additional reactions that prevent failure, the new schedule of reactions can be downloaded to the RTS.

CIRCA's State Space Planner builds reaction plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans [11]. To describe a domain to CIRCA, the user inputs a set of transition descriptions that implicitly define the set of reachable states. For example, Figure 3 illustrates several transitions used in the Puma domain. These transitions are of three types:

Action transitions represent actions performed by the RTS.
Temporal transitions represent the progression of time and continuous processes.
Event transitions represent world occurrences as instantaneous state changes.

The SSP plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. The SSP assigns to each reachable state either an action transition or `no-op`. Actions are selected to *preempt* transitions that lead to failure states and to drive the system towards states that satisfy as many goal propositions as possible. A planned action preempts a temporal transition when the action will definitely occur before the temporal transition could possibly occur. The assignment of actions determines the topology of the NFA (and so the set of reachable states): preemption of temporal transitions removes edges and assignment of actions adds them. System safety is guaranteed by planning action transitions that preempt *all* transitions to failure, making the failure state unreachable [11]. It is this ability to build plans that guarantee the correctness and timeliness of safety-preserving reactions that makes CIRCA suited to mission-critical applications in hard real-time domains.

The NFA is translated into a memoryless controller for downloading to the RTS. This is

```
#<TAP 10>
        Tests   : (AND (PART_IN_GRIPPER NIL) (EMERGENCY T))
        Acts    : push_emergency_button
        Max-per : 9984774
        Runtime : 2520010
#<TAP 9>
        Tests   : (AND
                    (PART_IN_GRIPPER NIL)
                    (EMERGENCY NIL)
                    (PART_ON_CONVEYOR T)
                    (NOT (TYPE_OF_CONVEYOR_PART SQUARE)))
        Acts    : pickup_unknown_part_from_conveyor
        Max-per : 12029856
        Runtime : 3540010
#<TAP 8>
        Tests   : (AND
                   (TYPE_OF_CONVEYOR_PART SQUARE)
                   (PART_IN_GRIPPER NIL)
                   (EMERGENCY NIL))
        Acts    : pickup_known_part_from_conveyor
        Max-per : 12029856
        Runtime : 3520010
```

Figure 4. Sample output from the TAP compiler.

done through a two-step process. First, the action assignments in the NFA are compiled into a set of *Test-Action Pairs* (TAPs). The tests are a set of boolean expressions that distinguish between states where a particular action is and is not to be executed. Each TAP's test expression is derived by examining all of the planned actions and finding a logical expression that distinguishes between the states in which the current TAP's action is planned and the states in which other actions are planned. Some sample TAPs for the Puma domain are given in Figure 4.

Eventually, the TAPs will be downloaded to the RTS to be executed. The RTS will loop over the set of TAPs, checking each test expression and executing the corresponding action if the test is satisfied. The tests consist only of sensing the agent's environment, rather than checking any internal memory, so the RTS is asynchronous and memoryless.

However, before the TAPs can be downloaded, they must be assembled into a loop that will meet all of the planned deadlines, captured as constraints on the maximum period of the TAPs (see Figure 4). This second phase of the translation process is done by the scheduler in the CSM. In this phase, CIRCA's scheduler verifies that all actions in the TAP loop will be executed quickly enough to preempt the transitions that the planner has determined need preempting. The tests and actions that the RTS can execute as part of its TAPs have associated worst-case execution times that are used to verify the schedule. If the scheduling

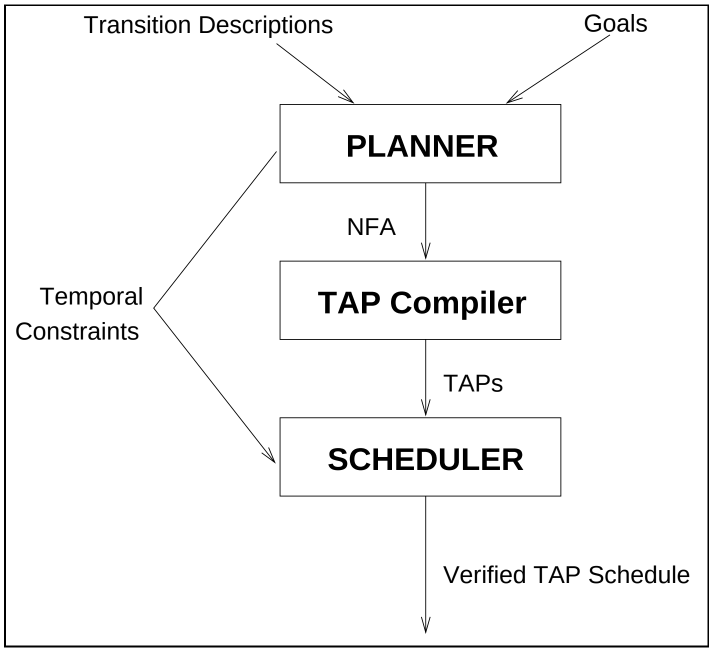


Figure 5. Summary of the CIRCA state space planning process.

does not succeed, the SSP will backtrack to revise the NFA, leading to a new set of TAPs and another scheduling attempt. The planning process is summarized in Figure 5.

3. Verifying State Space Plans

3.1. The CIRCA SSP

The CIRCA SSP automatically synthesizes timed discrete-event controllers for hard real-time applications. The input to the SSP is a description of a control problem in the form of environment dynamics (including uncontrollable processes and threats to system safety), actions available to the controller, and goals to be realized. The SSP returns a controller that is guaranteed to maintain the safety of the controlled system. The controller specifies what action should be taken for each reachable system state. The controller provides safety guarantees by meeting the timing requirements of the control problem; these timing requirements are inferred from the model of the uncontrollable processes that threaten the system. To determine that these timing requirements are met, our algorithm consults a model-checker for real-time automata. This model-checking is done on an incremental basis, as the controller is built.

For example, Figure 6 contains the transition descriptions for a simple UAV control problem. The transitions describe a problem in which a UAV is attempting to follow a normal flight path (hence the `*goals*` statement). However, at any time during its flight, the UAV might be tracked by enemy radar. Some time after the initial tracking, a surface-to-air missile (SAM) may be launched. If no countermeasures are taken, that SAM may destroy the UAV after at least a certain minimum amount of time has passed (e.g., the minimum flight time of the missile). The UAV has available to it some evasive maneuvers that will cause the SAM to miss the UAV, if the UAV initiates its maneuvers quickly enough. Also, since the maneuvers divert the UAV from its nominal trajectory, the UAV should end its evasive behavior whenever possible.

Figure 7 shows the state space resulting from a simple timed controller design that will preserve the safety of the UAV. In the initial state, labeled "State 17" and shown as a shaded oval, the UAV is on its normal trajectory and has no indication of a radar-guided missile tracking it. This is a desirable state, so the controller will make no effort to leave it. However, at any time, a radar threat could occur, moving the system into state 16. The controller will react to this threat by taking evasive action, and maintaining the evasive maneuvers until the missile has been avoided (i.e., until the system has entered state 24). At this time the threat has been neutralized, and the system is free to return to its normal flight path. This controller was automatically generated by CIRCA, and the state diagram was generated from CIRCA data structures by the daVinci program [5].

There are several important aspects to note about this example state space model, or finite automaton. First, note that the automaton contains loops: the UAV may be threatened by more than one missile, and will remain in (or re-start) evasive maneuvers as long as it is threatened. Second, observe that time is not an explicit part of the state representation. This is critical to the compact representation of looping plans; if we included time in the state representation, then loops would not occur and persistent reactive control against an unpredictable or adversarial world would explode the state space. Instead, our automaton neatly encodes the continously-reactive behavior of the UAV in a compact, efficient, and automatically-generated form. Of course, the transitions do have temporal semantics, as described in Figure 6.

The SSP's temporal model was carefully designed to support reasoning about system safety with only a minimal amount of temporal information, thus limiting the complexity of the automata model. We associate with each transition a set of bounds on the time (Δ) which the system must dwell in the transition's source state before the transition could possibly occur. The model includes four different types of transitions:

Temporal Transitions — Drawn as double arrows, temporal transitions represent uncertain processes that may lead to change, but only after at least some minimum amount of time has passed ($\Delta \geq min\Delta$). The only temporal transitions in our simple UAV example lead to failure, and are not shown in Figure 7 because the safety-preserving controller design makes failure unreachable.

Event Transitions — Drawn as single arrows, event transitions represent instantaneous transitions that are out of our control, and may happen any time their preconditions are satisfied. They are essentially the same as temporal transitions with a $min\Delta$ of zero.

Action Transitions — Drawn as dashed arrows, action transitions represent processes that are guaranteed to occur before the system has dwelled a certain amount of time in the source state. That is, action transitions will definitely occur before Δ reaches

```
(setf *goals* '((path normal)))

;; Radar-guided missile threats can occur at any time.
(make-instance 'event
        :name "radar_threat"
        :preconds '((radar_missile_tracking F))
        :postconds '((radar_missile_tracking T)))

;; You die if don't defeat a threat by 1200 time units.
(make-instance 'temporal
        :name "radar_threat_kills_you"
        :preconds '((radar_missile_tracking T))
        :postconds '((failure T))
        :min-delay 1200)

;; It takes no more than 10 time units to start evasives.
(make-instance 'action
        :name "begin_evasive"
        :preconds '((path normal))
        :postconds '((path evasive))
        :max-delay 10)

;; We defeat missile in between 250 and 400 time units.
(make-instance 'reliable-temporal
        :name "evade_radar_missile"
        :preconds '((radar_missile_tracking T)
                    (path evasive))
        :postconds '((radar_missile_tracking F))
        :delay (make-range 250 400))

;; It takes no more than 10 time units to end evasives.
(make-instance 'action
        :name "end_evasive"
        :preconds '((path evasive))
        :postconds '((path normal))
        :max-delay 10)
```

Figure 6. A simple domain description for a UAV threatened by radar-guided missiles.

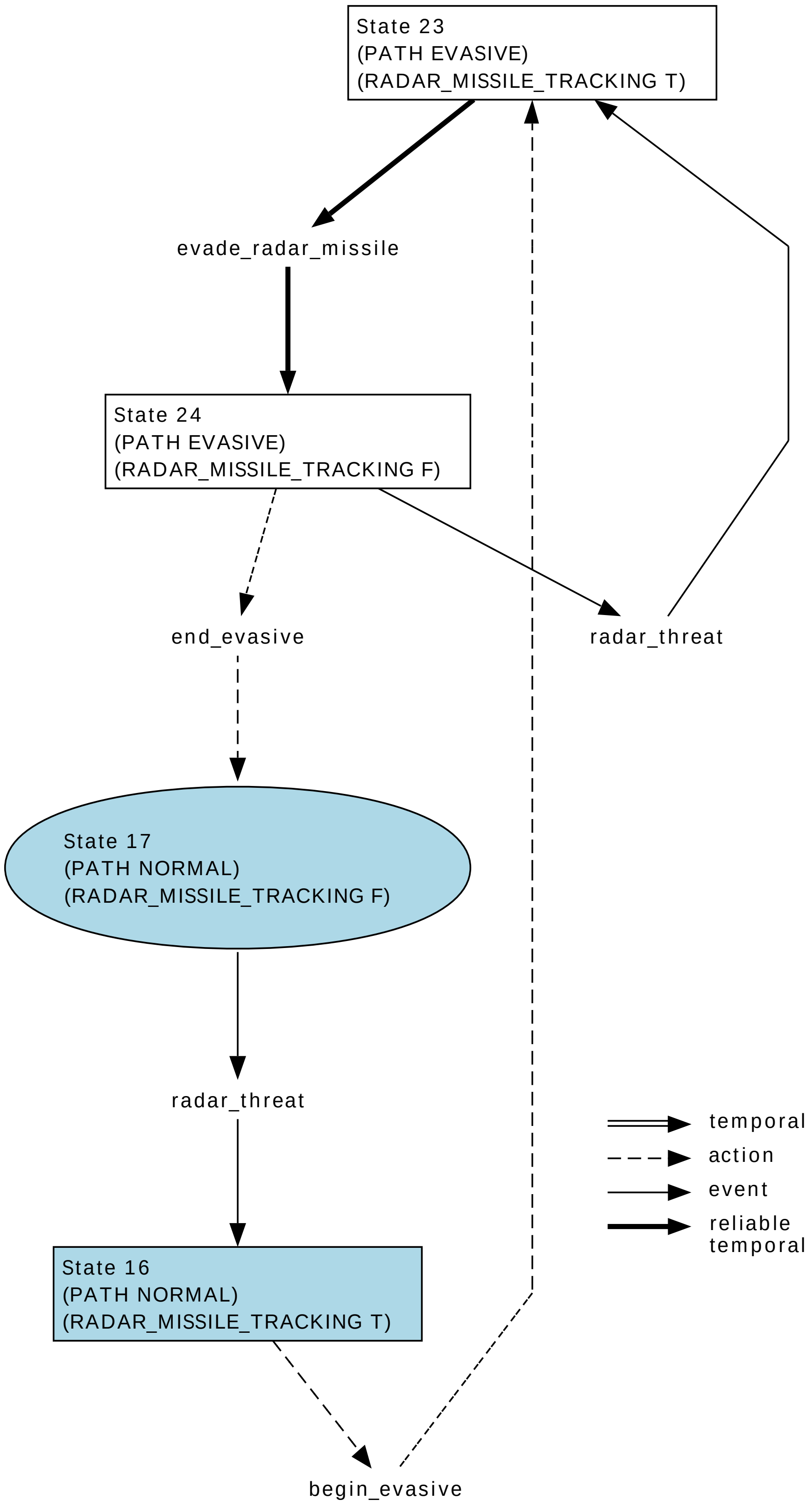


Figure 7. Simple UAV controller for evading radar-guided SAM threats.

an upper bound $max\Delta$.

Reliable Temporal Transitions — Drawn as bold single arrows, reliable temporal transitions represent processes that are guaranteed to occur, if given enough time. They have both lower and upper bounds on the dwell time the system must stay in the source state before the reliable temporal transition will occur ($min\Delta < \Delta < max\Delta$).

Using this information, the SSP reasons about one key temporal relationship: *preemption*. A transition t is preempted iff some other transition u from the same state must definitely occur before t could possibly occur. In other words, t is preempted iff $max\Delta(u) < min\Delta(t)$. In our UAV example, the `radar_threat_kills_you` transition is preempted in state 16 by the action transition `begin_evasive`.

Preemption is the key temporal relationship in CIRCA models because it allows the SSP to build discrete event controllers that make certain parts of the potential system state space unreachable. By making all potential failure states unreachable, the SSP can build plans (controllers) that are guaranteed to keep the system safe, while also pursuing other less-critical goals. The goal of plan verification, discussed in the next section, is to prove that the preemptions CIRCA has planned will in fact hold true for all possible future world "trajectories" (i.e., paths through the reachable states).

Note that the `begin_evasive` action does not actually disable the `radar_threat_kills_you` transition: it simply begins the process of defeating the threat, which is represented by the reliable temporal transition `evade_radar_missile`. So if we were to draw the temporal transitions to failure (TTFs) in the graph of Figure 7, we'd see that the `radar_threat_kills_you` TTF is actually preempted out of both state 16 and the subsequent state 23. This is called a *dependent temporal chain*, because the amount of time left to preempt the TTF in state 23 is not the original minimum dwell time (as it was in state 16), but the original $min\Delta$ minus however much time the system may have dwelled in state 16 before transitioning to state 23. Since CIRCA reasons about worst-case circumstances, we can assume that that dwell time, in the worst case, is equivalent to the upper bound dwell time ($max\Delta$) imposed by the planned action `begin_evasive`, so that the new $min\Delta$ in state 23 is actually $1200 - 10 = 1190$.

Thus our temporal model is actually non-Markovian: the temporal semantics of the TTF out of state 23 depend on the path the system takes to get there. Naturally, this complicates the process of reasoning about the temporal model, and motivates our use of model checking to verify the required TTF-preemption properties.

3.2. Model Checking for Plan Verification

In order to verify that the CIRCA SSP's plans are safe, we must project what will happen when they are executed. We must determine whether the actions we have planned do, in fact, preempt all possible transitions to failure. To do so, we use techniques developed in the computer-aided verification research community; specifically we use techniques for verifying properties of *timed automata* [1].

A naive algorithm for CIRCA plan verification is easy to propose: start at the initial state(s), find all the possible successor states, and repeat. If you ever enter a failure state, the verification has failed.

The problem with this algorithm is hidden in the definition of system state. To determine the possible successor states, we must know how long transitions have been enabled. For example, to determine at state 23 whether `radar_threat_kills_you` happens before or after `evade_radar_missile`, we must know whether the former transition has been active for 1200 time units before the latter has been active for 400 (see Figure 6).

Imagine that each transition has associated with it a timer, or "clock." When the transition is enabled, that clock is reset to zero and started. When the transition is disabled, that clock is turned off. Whenever that clock goes over the lower bound on the corresponding transition, the transition may occur; the transition is guaranteed to occur before the upper bound on the transition (unless some other transition intervenes).

Thus we can characterize the full state of the controlled system by the full set of feature values and a vector of artificial clock values. For example:

```
flight_path = evasive
radar_missile_tracking = true
clock(evade_radar_missile) = 40
clock(radar_threat_kills_you) = 700
```

By comparing this state against the problem definition given in Figure 6, you may readily see that this state is safe. `radar_threat_kills_you` cannot take place for 500 more time units, by which time `evade_radar_missile` will have preempted it.

Unfortunately, the verification problem, as naively framed, is not practically solvable. Since the clocks are integer-valued,[1] the set of system states is infinitely large. However, the set of interesting values is less than infinite, since there are only a finite number of decisions that need be made. For example, all values of `clock(radar_threat_kills_you)` that are over 1200 are equivalent. However, the number of relevant states may still be very large.

3.2.1. Timed Automata Representation

Fortunately, researchers in computer-aided verification have found ways to compactly represent states like this for a class of finite state machines called *timed automata* [1]. Timed automata differ in a few minor ways from SSP state machines, but SSP state machines can be translated into timed automata. Timed automata states are composed of a *location* (corresponding to an SSP state, or feature vector) and a *clock-interpretation, or vector of clock values.* All of the clocks increment synchronously, but can be independently reset to zero by selected transitions. Transitions themselves are instantaneous. Temporal constraints in timed automata take two forms: transition *guard expressions* that must be true to enable a transition, and *state invariant* expressions that must be true all the time

[1]Although time is continuous, it may be discretized without loss of accuracy for any verification problem.

the system remains in a particular state.

Mapping an SSP state space model into a timed automaton is a fairly simple matter of assigning different clocks to different CIRCA transitions and translating the CIRCA transition timing constraints into timed automaton clock constraints. Once this translation is complete, the timed automaton model can be passed to our model-checking code, the Real-Time Analysis (RTA) module, to determine whether failure is reachable and, if so, what path of transitions leads to failure (to guide CIRCA's intelligent backjumping).

Figure 8a illustrates the RTA timed automaton that corresponds to CIRCA's solution to the UAV example of Figure 7. Briefly, the automatic translation process involves mapping each type of SSP state space transition, as follows:

Temporal Transitions — Temporal transitions require the system to dwell in a state for a certain amount of time before the transition may occur. This corresponds exactly to a transition guard expression in RTA. Thus temporal edges are each assigned a clock, and have guard expressions constraining the value of that clock to be greater than the temporal transition's minimum delay. The clock is reset by all edges entering the source state of the temporal edge, if that edge does not come from a state in which the same temporal is enabled.

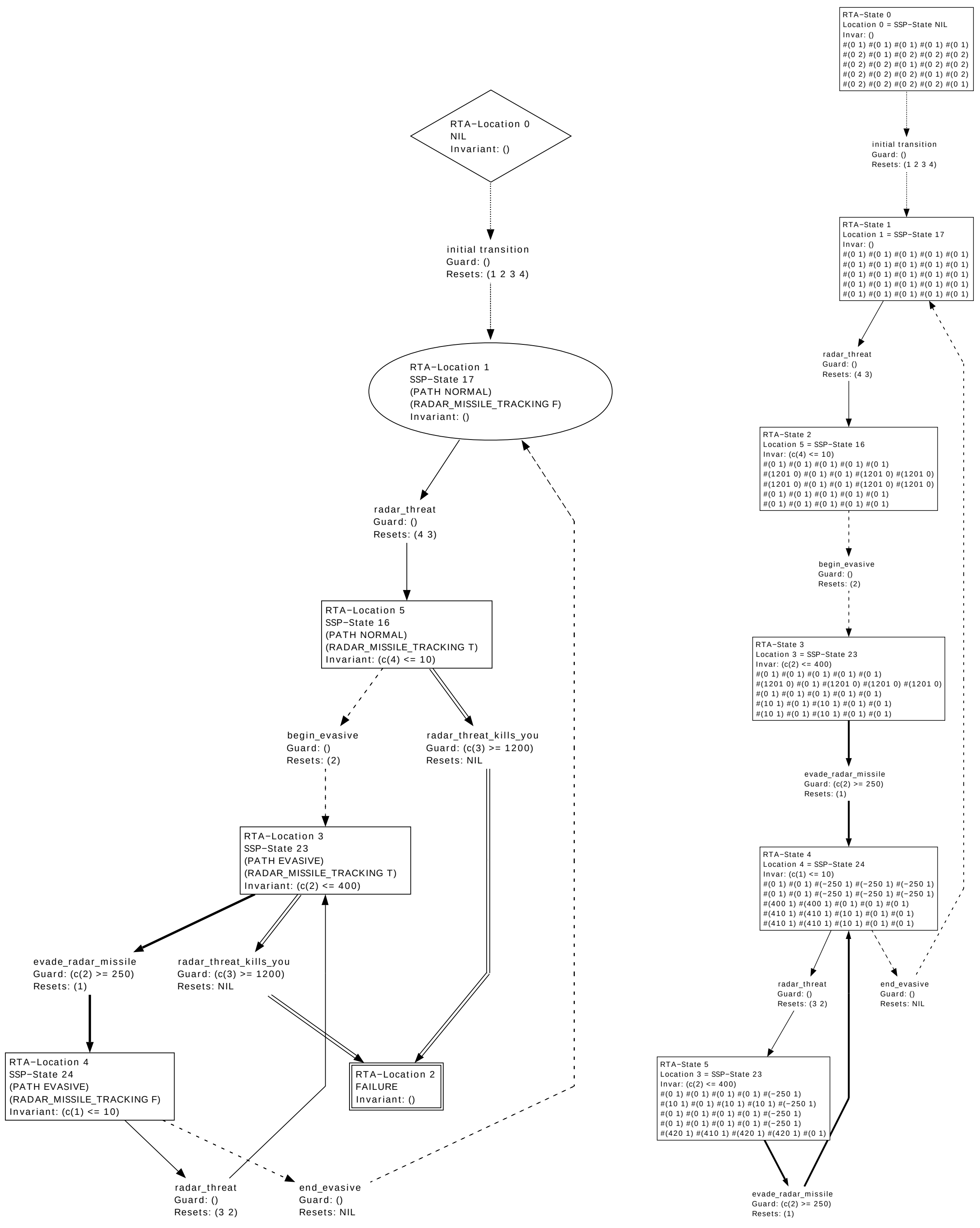
Event Transitions — Because event transitions can occur at any time, they have no associated clocks and are simply unrestricted edges in the RTA graph.

Action Transitions — Recall that action transitions place an upper bound on the time the system may dwell in the transition's source state before it necessarily will move to the transition's sink state. In our RTA model, this corresponds to an upper bound state invariant expression. Each instance of an action transition (action edge) is assigned a new clock. The clock is reset by all edges entering the source state of the action edge, if that edge does not come from a state in which the same action is enabled. The action edge itself has no guarding clock constraints; instead, the action edge's upper bound is expressed as an invariant in the edge's source state.

Reliable Temporal Transitions — Reliable temporals combine the lower-bound and upper-bound timing constraints of temporals and actions, so their RTA mapping uses transition guards to represent the lower bounds and state invariants to represent the upper bounds.

3.2.2. Efficient Model Checking

The critical concept for taming the complexity of timed automaton verification is an equivalence relation ("region equivalence") between system states [1]. This equivalence relation makes use of the intuition that all values for a given clock are equivalent above a maximum value (the largest constant the clock is ever compared to). Furthermore, since we are only concerned with the reachability of various states, the actual values of different clocks in a state are not as important as their *relative* values. Because the clocks are all notionally incremented at the same rate, the relationships between the clock values upon entry to a state is sufficient to determine which outgoing transitions are possible: a clock



(a) Input finite automata. (b) Clock-zone expansion.

Figure 8. The input model and clock zone analysis for the example UAV domain.

that is behind another cannot catch up (within a state). Based on this equivalence relation, it can be shown that any timed automaton (SSP plan) has only a finite number of states.[2] Therefore, the problem of determining reachability (SSP plan verification) is decidable.

A further optimization is possible, to make verification practical. The key intuition behind this optimization is that all reachability questions hinge on pairwise comparisons between clock values. In order to determine whether or not one transition can occur, we compare a single clock against a constant. To determine whether one transition occurs before another, we only need to determine which will reach its associated constant first. To answer this question, we only need to know the *difference* between pairs of clock values (since the clock values increase at the same rate).

Therefore, we can compactly represent clock regions using a *difference-bound matrix* [4] whose entries represent bounds on the difference between pairs of clocks and between single clocks and a dummy clock whose value is always zero. Difference-bound matrices have two advantages. First, they provide a compact representation for equivalence classes of clock-states in timed automata. Second, they also have a canonical form, derived using any standard all-pairs shortest-path algorithm [4]. Putting the associated difference-bound matrices into canonical form makes it easy to determine when two automaton states are equivalent. Recognizing equivalent states is, in turn, necessary in order for reachability search to terminate.

Figure 8b illustrates the reachability verification of the SSP plan given in Figure 6, optimized by the use of difference-bound matrices. Space limits preclude us from describing the difference bound notation in detail. However, a simple examination of Figure 8b shows one notable aspect of the RTA verification: there are two RTA states (3 and 5) that correspond to the SSP state 23. That is, the RTA algorithm has recognized the distinction between the two routes into SSP state 23 (see Figure 7) as being a temporally significant difference. The temporal transition to failure from state 23 will have different amounts of time left on its clock depending on whether we enter from state 16, where it was already enabled, or state 24, where it was not enabled (see Figure 8a, RTA-locations 5 and 4). Thus the RTA algorithm is unrolling the important paths through dependent temporal chains, checking reachability of failure by removing the original non-Markovian temporal semantics.

3.3. Multi-Model Verification

The original model-checking interface captured the connectivity and temporal restrictions of the SSP's state space graph and sent it Kronos for reachability verification, to make sure that failure is not reachable. One weakness with this approach is that it relies on our SSP implementation to reason effectively about the "execution semantics" of the actual RTS TAP plans that will be generated by the SSP. That is, we relied on our SSP code to correctly map the SSP state space model into TAPs: we were verifying the SSP state space

[2]More precisely, there are only a finite number of state equivalence classes, and state equivalence classes are sufficient to determine reachability.

model, which was then converted into TAPs *after* the verification. The alternative is to actually map TAPs directly into Kronos simulation/verification models, so that Kronos can reason about the interactions of the planned TAPs directly with the SSP's world model, and essentially verify that the SSP has correctly built both its state space model *and* the TAP plan. This approach uses multiple concurrent state machine models to represent all exogeneous transitions and TAPs. This "multi-model" analysis approach will verify that the TAP-based RTS implementation of a CIRCA plan will actually support all the performance guarantees that the SSP believes it will. This approach also offers the opportunity to take advantage of Kronos' efficient state-space cross-product behavior: the model checker is good at deriving minimal cross-products that avoid enumerating unnecessary system states.

Unfortunately, once we implemented the necessary interfaces translating the CIRCA TAP plans into multi-model verification problems, we encountered significant scalability problems with Kronos. When it detects a verification failure (because some failure state is reachable in a tentative SSP plan), Kronos is unable to efficiently produce an example state-space path leading from the start state to failure. Apparently because Kronos does some pre-enumeration of the combinatorially-explosive cross product of multiple state machines, it does not scale well on these problems.

As a result, we also be began extending our customized RTA verifier to the multi-model case. The multi-model RTA (MMRTA) system has been prototyped and passed initial tests, but we suspended the remaining work (prior to ANTS funding) to focus on the system demonstrations and final report. The remaining MMRTA work, which is not too complicated, is to automate the translation of a CIRCA SSP verification problem into the multi-model RTA input format.

Note that the MMRTA will inherently reason about "ghosting," which is our term for actions that were planned for some state A, but occur in some different state B, because of an exogenous transition that moves the world from A to B after the RTS has sensed A, but before the RTS has executed the action. The MMRTA will automatically reason about these situations since it will expand the state space including possible orderings of TAP activities and exogenous transitions. While we expect MMRTA to inherently detect problem situations where ghosted actions lead to failures, it will take some additional work to modify the SSP to use that information in guiding its search/backjumping.

3.4. Related Work

The CIRCA SSP is a reaction planner, and thus has much in common with work on reactive systems in AI and control theory. CIRCA is unusual in two ways: it automatically synthesizes, or plans, its reactions, and it provides performance guarantees through the methods of hard real time.

In independently-developed work, Asarin, Maler, Pneuli and Sifakis [2, 9] developed a game-theoretic method of synthesizing real-time controllers. They view the problem as trying to "force a win" against the environment, by guaranteeing that all traces of system execution will pass through (avoid) a set of desirable (undesirable) states. Their method is very similar to ours, but their work stopped at the development of the algorithm and derivation of complexity bounds; it was never implemented. Our work concentrates on the implementation aspects of this problem and on making it computationally practical.

Kabanza, et. al. have developed work [6, 7] very similar to ours in intention. Their early work (fully presented in [7]) is similar to the original CIRCA State Space Planner work, but does not take into account metric temporal information. Later work [6], extends the original framework by incorporating metric time, but does so by effectively imposing a system-wide clock and progressing the controller one "tick" at a time. In control problems with widely varying time constants, this approach will lead to an explosion of states; we have adopted model-checking techniques that minimize this state explosion.

Markov Decision Processes and Partially-Observable Markov Decision Processes provide a theoretical basis for planning and action that is similar to discrete control theory, but they place the accent on uncertainty [3]. CIRCA simply represents uncertainty through nondeterminism: CIRCA transitions may have alternative outcomes; uncontrollable events may or may not occur; etc. The SSP techniques discussed in this paper do not attempt to reason about quantified measures of uncertainty, they make the worst-case assumption: "anything bad that can happen will happen." However, there has been some preliminary work on developing a probability model for CIRCA, to permit principled model-pruning decisions [8].

4. The Probabilistic State Space Planner

When faced with overconstrained problems, the CSM described above is unable to find a safety-preserving plan, and it returns failure to the AMP. At that time, the AMP is designed to relax one or of the constraints on the CSM, perhaps by redesigning the problem configuration to include a faster action transition, or modifying a goal. Our University of Michigan colleagues have been working on a new version of the SSP that uses probabilistic information to guide the system in considering the most-probable states first. As a result, the Probabilistic State Space Planner (PSSP) is able to build partial plans that are only probabilistically safe, because they only consider those states whose likelihood is above a given threshold. This type of partial plan is useful in situations where the domain is highly challenging, and the CSM may have only a limited amount of computation time to come up with a suitable control plan for the next phase of CIRCA operations. Or, the RTS itself may have limited abilities to sense and react to the world, and these may preclude the CSM from generating a complete real-time controller, and require some performance tradeoffs. In any case, by explicitly pruning least-probable areas of the state space, the PSSP approach allows CIRCA to optimize its allocation of reactive resources. Moreover, if additional planning-time resources are available and the RTS is the limiting factor, the system can build contingency plans to handle pruned areas of the state space and swap those plans in when the pruned, less-likely situations arise.

Michigan's progress on this system are described in the attached technical paper, Appendix **??**. The results of their work are illustrated in the demonstrations described in Section 6.

5. The Adaptive Mission Planner

Our investigation into the CIRCA Adaptive Mission Planner (AMP) pursued several research branches before ending prematurely due to an unexpected budget reduction. We began by clarifying the role of the AMP, and designing the problems it must solve to control and guide the CSM. Since the AMP requirements are similar to existing AI planners (much more so than the CSM), we explored the possibility of using an existing planner, with enhancements, to fill the AMP role. As described below, we investigated the SIPE-2 and Prodigy planners in depth, and began prototyping experiments.

5.1. Requirements for the AMP

Based on prior experience with the CIRCA architecture and our scenario designs for the UCAV demonstration domain, we developed a set of general functional requirements for the AMP. We want the AMP to:

Project — The system must be able to reason about potential future states of the world, to be a true projective planner (rather than simply reacting to the current world state).

Search — The system must maintain an explicit representation of planning decisions for possible re-evaluation and change.

Build Phases for the CSM — The primary AMP responsibility is to divide up the overall problem state space into a series of overlapping regions of competency, or phases. The CSM will then build real-time reactive control plans for each phase.

Modify Phases when the CSM Fails — If the CSM fails to generate a safe controller for a given subproblem, the AMP must modify its plans to reduce the complexity of that phase.

Incorporate Changes During Execution — Because the AMP is the long-term planner in CIRCA, it must be able to handle deviations from its coarse plan as the situation progresses.

Set up Execution-Time Sentinels/Monitors — The AMP must be able to automatically monitor its own performance to detect deviations from the plan.

Plan for Domain Resources — The AMP must build plans that account for domain resources such as fuel.

Manage Reasoning Resources — The AMP must control the CIRCA reasoning process itself, so that the CSM builds controllers in a timely fashion; if the phase problems sent to the CSM are badly formed, the CSM may never return success or failure. The AMP must monitor CSM performance and adjust its plans to meet the soft real-time deadlines imposed by the domain.

Incorporate New Operators — The SA-CIRCA architecture calls for the ability to synthesize new code modules in response to changing performance results; the AMP

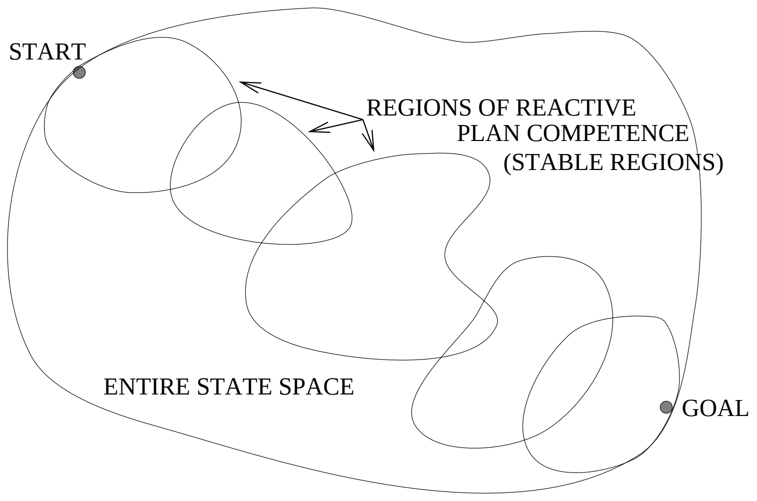


Figure 9. Conceptual view of multiple reactive controllers.

must be able to reason about new, dynamically-created domain operators that can be used to improve performance.

Optional functional requirements include abilities to:

Plan for Multiple Agents — Distributed applications of CIRCA will require the AMP to communicate and coordinate with other agents to effectively manage roles, responsibilities, and closely-coordinated behaviors.

Direct Execution — The AMP should direct the overall execution of real-time control plans by managing the plan cache in the RTS; this requires interleaving execution and planning.

Synthesize New Operators — In the full SA-CIRCA design, one way the AMP may respond to performance deviations is to actually design and synthesize new operators that give it missing domain capabilities.

5.2. Interface to the CSM

While developing these requirements, it became clear that there is a fundamental mismatch between the output of a standard planner (a plan), and the input we require for the CSM. To understand this problem, recall that the AMP is notionally responsible for dividing up the overall problem state space into a series of overlapping regions of competency, as shown in Figure 9. The CSM will then build real-time reactive control plans for each region of competency. If the CSM is unable to build a feasible plan for a particular region, then the AMP must reorganize/replan to develop a different breakdown of the state space, or possibly give up on some of its goals. In essence, the AMP is responsible for breaking up a large problem into smaller problems for the CSM to solve, and adjusting that breakup to compensate for difficulties the CSM may encounter.

The interface between the AMP and CSM is non-obvious because the CSM requires as input a “problem configuration” consisting of a set of temporal and event transitions (describing the domain events and processes), a set of action transitions (describing control actions that can be taken), a set of starting states, and a set of goal states (or conditions). In contrast, a traditional planner's output is a plan consisting of a set of primitive operators, possibly with associated parameter bindings. We want the AMP to be able to search for suitable sequences of problem configurations, potentially varying any and all aspects of the configurations (e.g., adding new action transitions when automatic code synthesis provides custom-built primitives, removing temporal transitions with low probabilities, varying action transition delay parameters, etc.).

How can we map an AMP's operator/parameter plan formulation into the complex structure of a CSM problem configuration?

The unique, very powerful approach we are currently pursuing involves extending the domain representation of the AMP to include explicit representation of the problem configuration elements. That is, rather than have the AMP only think about the domain, we have the AMP also think explicitly, in its search space, about the CSM problem configurations. The AMP will plan with both domain operators and "configuration operators" that modify the problems sent to the CSM.

For example, high-level UCAV domain activities include aircraft trajectories, missile threats, etc. These will be reflected in the AMP representation as "domain operators." The AMP will use these domain operators to build a coarse-grain plan through the domain, specifying, for example, that the mission will consist of an ingress phase, an attack phase, and an egress phase. The AMP will use the CSM to develop more detailed controllers to invoke during each of these phases. To task the CSM with problem configurations for each of these phases, the AMP must also decide, for example, which of several reactive evasive-maneuver actions should be made available to the CSM for use in a particular phase. The AMP will have "configuration operators" that can add or remove those action transitions from the problem configuration sent to the CSM. The configuration operators will not directly affect the AMP's domain model, because they are below the level of detail that the AMP is projecting the world. However, by changing the details of the problem configurations sent to the CSM, these configuration operators may have significant impact on the CSM's results.

Now suppose the CSM is unable to produce a safe controller for the given problem configuration, and the AMP must make some change. Because the configuration operators that built the CSM problem configuration are treated just like domain operators, the AMP's search machinery works on them just like other operators. So *the AMP can search over the set of possible problem configurations* by searching over its configuration operator choices. In other words, the configuration operators bring the CSM problem configurations under the direct control of the AMP search machinery, which is exactly what we want: the AMP can now reason explicitly about how it is breaking up larger problems into smaller ones, and how the smaller problems are structured.

Feature	SIPE 2	Prodigy	PRS	MACBeth
Projection	√	√		√
Search	√	√		√
Build CSM Phases				
Modify on CSM Failure	√	√	√	√
Handle Execution Failures	√	√	√	√
Set up Monitors		√	√	
Plan for Domain Resources	√			√
Manage Reasoning Resources				
Incorporate New Operators			√	
Multi Agent Planning				√
Direct Execution	√	√	√	
Synthesize New Operators				

Figure 10. Comparing existing planners against AMP requirements, we find no perfect match.

5.3. Potential Starting Points

Since the AMP requirements are similar to existing AI planners, we explored the possibility of using an existing planner, with enhancements, to fill the AMP role. Based on our knowledge of the planning literature, we selected several different larger-scale planning systems (that have been relatively well-developed and used) to evaluate against the AMP requirements. Figure 10 summarizes our comparison against SIPE-2, Prodigy, PRS, and MACBeth.

Based on this comparison, we tried to build initial AMP functionality in SIPE-2, as described in the next section. When this effort failed, we moved on to Prodigy, and the project budget cut occurred shortly after our investigation into Prodigy was concluded.

PRS was considered, but because it is fundamentally a reactive system and not a projective planner, it is most useful for "hand-coding" or programming AMP behaviors. PRS does not work by building a plan of future actions and revising it as execution occurs; instead, PRS repeatedly selects only the next action to take. This makes it very flexible and responsive to changing circumstances, but subject to the short-sightedness of reactivity. For the AMP, this means PRS is useful for quickly building known functions and behaviors (as was done in the original CIRCA implementation [10]), but not as a long-term solution capable of model-based adaptivity.

MACBeth is a constraint-based planner using hierarchical task networks, developed at Honeywell for tasking and control of robot and UAV systems. MACBeth does not have the long history of development and applications shared by the other systems we evaluated, but it has powerful resource reasoning based on constraints. Major problems with MACBeth include its lack of interleaved execution and monitoring, and the difficulty of incorporating new planning operators in the code.

5.4. Using SIPE-2 for the AMP

SRI's SIPE-2 planner is a well-established system that appears to provide many of the features we require, and it has been used in several commercial real-world applications. The government funded SIPE's development, and Rome Labs already has rights to the software. We obtained object-code versions of the system directly from SIPE-2's primary developer, Dr. David Wilkins at SRI.

SIPE-2 is an HTN-style planner, meaning that it builds plans by joining together pieces of hierarchical task networks. The documentation indicates that SIPE-2 supports reasoning about domain resources and parallel actions (nonlinear plans). The planner works by repeatedly selecting problems with the current partial plan and resolving them, using code modules called critics. These plan critics check the global constraint network for inconsistencies, identifying resource conflicts and problematic interactions between unordered actions. The planner can represent context-dependent effects, supports interactive or automated search for solutions, and has some replanning functionality. The system also includes a GUI and a few small demonstration domains.

Unfortunately, while the outward appearance gives a good match to our needs, we found it impractical to work with SIPE-2. Initially, we encountered numerous small but fatal bugs in the GUI, interaction metaphors, and demonstration scripts. We engaged in several iterations of bug-reports and patches with SRI, but the SIPE-2 user interface is particularly clumsy and the complexity of the system's behavior is a challenge. SRI personnel including Dr. David Wilkins and Dr. Karen Myers were very helpful in getting us up to speed.

Once we got the system working on its example domains, we implemented very preliminary UAV-like domains and generated simple plans in these domains. In one of the SRI demonstration domains, we were also able to get the replanning functions to operate, although only on a very simple replanning problem. Efforts to employ similar replanning in our UAV domains were futile. Expected SIPE-2 functions such as resource reasoning, replanning, and even heuristic search guidance are undocumented, more difficult to use than expected, or possibly even unusable without the planner's source code. In addition, it became clear that the SIPE-2 API is not sufficient for our needs, being largely undocumented. Brittleness in the system made it virtually impossible to develop domains that varied significantly from the examples included with the distribution. Finally, after several months of effort, we concluded that it was not feasible to use SIPE-2 without the source code, and we abandoned the effort.

5.5. Using Prodigy for the AMP

We re-examined our AMP requirements and decided to investigate the Prodigy planner from CMU to see if it would be easier to use for an AMP prototype. Prodigy is freely available for research, and we already had the system, source code, and experts on-site. Prodigy is a goal regression planner that backchains from set of initial goals, building plans that always consist of a totally ordered sequence of operators. Prodigy operates in a nonlinear fashion, choosing which goal or subgoal to work on based on a variable heuristic formed from control rules. Prodigy can consider operators with limited types of conditional effects, and it supports the notion of "applying" an operator to provide more complex forms of forward operator inference that cannot be regressed over.

Preliminary results with Prodigy were much better than with SIPE-2: we implemented simple domains, debugged several problems with the planner's source code in fairly short order, and identified several required enhancements that were only feasible because we had the source code. Positive aspects of Prodigy include:

- Simple operational concept, well documented.
- Source code available, fairly well written.
- Useful directed backjumping and replanning.
- Simple support for interleaved planning and execution.
- In-house HTC expertise (Dr. Karen Haigh).

In the process of building and testing a small UAV demonstration domain for Prodigy, we encountered several problems that indicate major weaknesses, for our purposes. These include:

- No true parallel actions.
- Little or no effective resource reasoning.
- No way to handle goals of avoidance (e.g., don't run out of fuel).
- No direct control of search backtracking.

5.5.1. Goals of Avoidance

One of the key aspects of the UAV planning domain that we had difficulty encoding into Prodigy is the notion that the system must not produce plans that use up more fuel than the aircraft has. We'd like to express a "goal of avoidance:" always avoid the situation where fuel is zero or less. Note this is different from standard goals of achievement ("get to the goal"), even with negation. Standard achievement goals are not true during some early part of the plan, and the plan makes them true. The plan is valid whether the goal is true or not. With a goal of avoidance, however, we must produce plans in which the undesirable situation *never* occurs: we must never run out of fuel. If we simply tell Prodigy that we'd like to have fuel greater than zero, it will happily run the fuel below zero and then try to add a new operator that will restore the goal (perhaps by refueling). Of course, it is not satisfactory to have a plan that runs out of fuel, and then tries to refuel.

We need a new type of goal, for which any violation is treated as a planning failure and leads to backtracking/replanning. Implementing this type of goal requires fundamental changes to the planning behavior of Prodigy: it must check all of these goals of avoidance each time it projects a new state, and initiate backtracking if any are violated. We made this change to the Prodigy code, and successfully demonstrated the new behavior. As a result, we can now properly handle certain types of avoidance goals.

5.5.2. Motivating the Use of Problem Configuration Operators

Perhaps the most interesting and planner-independent observation made during our prototyping efforts was the challenge of "motivating" the Prodigy planner to include the appropriate CSM problem-configuration operators into the mission plan. Prodigy plans actions that achieve a subgoal; the problem with configuration operators is that they do not directly achieve any subgoal, but rather configure the description of the problem for the CSM to examine. For example, in a phase of the mission where a heat-seeking missile is a likely threat, we have several configuration operators that should be included in the AMP plan so that, when that segment of the AMP plan is passed to the CSM for controller synthesis, the CSM accounts for that missile threat. We'd essentially like all of the applicable configuration operators to be applied by default, and then have the AMP's search mechanisms explicitly decide which configuration operators to remove if necessary. For example, if the CSM is unable to build a feasible plan with the missile threat accounted for, the AMP can either replan the route or decide to not worry much about the missile threat, omitting those configuration operators from the plan and thus "assuaging" the CSM's conservative worries.

We have not yet solved the problem of motivating the addition of configuration operators; it appears that this functionality should also be added to the core planning behavior of the algorithm, but the exact definition and structure of the modification is not yet complete. We face the difficulty that making a very large planning system like Prodigy do new things is very challenging.

6. Demonstrations

As noted above, we guided the SA-CIRCA project design and development tasks by aiming towards scenarios in which SA-CIRCA controls a UAV engaged in combat missions. We hoped to obtain a ready-made simulation of UAV systems from Wright Patterson AFB, but licensing issues made it impossible for them to deliver the software to us. As a result, we were forced to enhance an existing CIRCA-compatible flight simulation to provide simple combat-oriented simulation functions.

6.1. The UAV Simulation

Leveraging prior work on flying UAVs with CIRCA (both at Michigan and Honeywell), we began with a demonstration system capable of simulating a simple F-16 aircraft model flying over high-resolution terrain. The simulation only ran on high-powered Silicon Graphics computers. The CIRCA controller was interfaced at a fairly high level, providing waypoints and other discrete commands (e.g., deploy flaps) to the simulated aircraft. The simulation itself provided a simple autopilot to fly towards waypoints, and an auto-throttle function to maintain appropriate speeds.

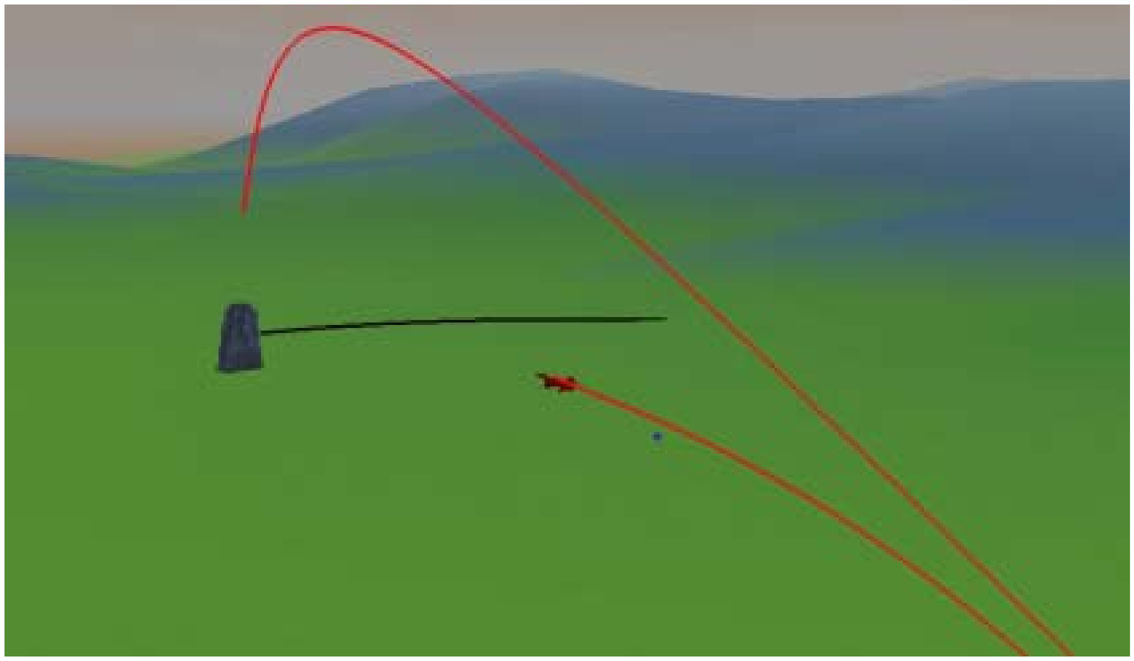


Figure 11. The demonstration simulation illustrates an SA-CIRCA-controlled aircraft responding to attacks with evasive maneuvers, flares, chaff, and counterattacks.

For the SA-CIRCA demonstrations, the simulator was extensively modified to enable:

- Deployment of chaff and flares.
- Automated evasive maneuvers.
- Ground-launched IR-guided and radar-guided missiles that track the aircraft and are distracted by flares or chaff, respectively.
- Smoke trails for aircraft and missiles, to provide persistent trajectory display.
- Projected future waypoint/path display.
- Ground-based threat installations (SAM sites and IR launch sites).
- Air-launched missiles that track ground targets.
- Low-resolution graphics display to run on Michigan's slower machines.
- A rebuilt CIRCA command interface to allow easy expansion and provide access to the new control functions (e.g., flare deployment).

Figure 11 shows a screengrab from one simulation run, in which the UAV has been threatened by a surface-to-air missile (seen as a black line curving towards the aircraft) and it is responding with flares (one of which is seen falling below the flight path) and a counter-strike missile (seen arcing down onto the surface installation).

6.2. Demonstrations of the Architecture

We encoded SA-CIRCA domain models for several variant UCAV demo scenarios, involving radar-guided and IR-guided missile threats and several actions that the aircraft must take in response to those threats (chaff, flares, evasive maneuvers), as well as target strike goals. For example, Figure 6 (on page 9) illustrates one fairly simple domain encoding in which only radar-guided missiles and evasive maneuvers are considered. Using these domains, we generated a variety of TAP controllers capable of responding to different spectrums of threats.

These controllers were tested in the simulation, and several of the runs were videotaped for presentation at the program review in July, 1999. Demonstrations showed the controllers responding in a timely fashion to threats, and also communicating with high-level replanning functions when non-critical goals were threatened. For example, in one scenario the aircraft responds to a missile threat by evasive maneuvers and chaff, but the resulting divergence from the planned trajectory means that the aircraft will miss its assigned "time on target" if it resumes the prior path. So, recognizing that this goal is threatened, the system invokes a path planner on-the-fly and derives a new trajectory that follows a significantly different path (actually passing through a different river valley) to make up for the lost time.

Our Michigan teammates used the same simulator, but modified the domain models to incorporate probabilistic information suited to their demonstration scenarios, described below.

6.3. Demonstrations of the PSSP

The goal of the PSSP demonstrations was to show that the probability information can be used, in overconstrained domains, to shed "the right stuff:" the lowest-probability states and contingencies. And, to show that explicit knowledge of what has been shed can be used to build separate contingency plans that detect and react when the improbable shed states actually occur.

The underlying assumption is that the RTS isn't capable of making all guarantees. To keep the demo/simulation simple, there are only 2 temporal transitions to failure (TTFs), associated with the two types of missiles. More realistically, we would assume that our plane would have many other failure modes to preempt. Here we assume that there are enough other important TTFs being addressed that the remaining resources are insufficient to guarantee reactions for both radar-guided and IR-guided missiles.

Figure 12 shows the probability rate functions used for the PSSP demos. Both IR and radar threats are equally likely from the states in which they are enabled. However, IR threats are enabled only when the aircraft is at low altitude, and radar threats only when it is at high altitude. Furthermore, IR threats are more likely to kill you if they occur. However, the aircraft is modeled as only occasionally (and, in fact, non-volitionally) moving to low altitude, and then always returning to high altitude using the `climb`

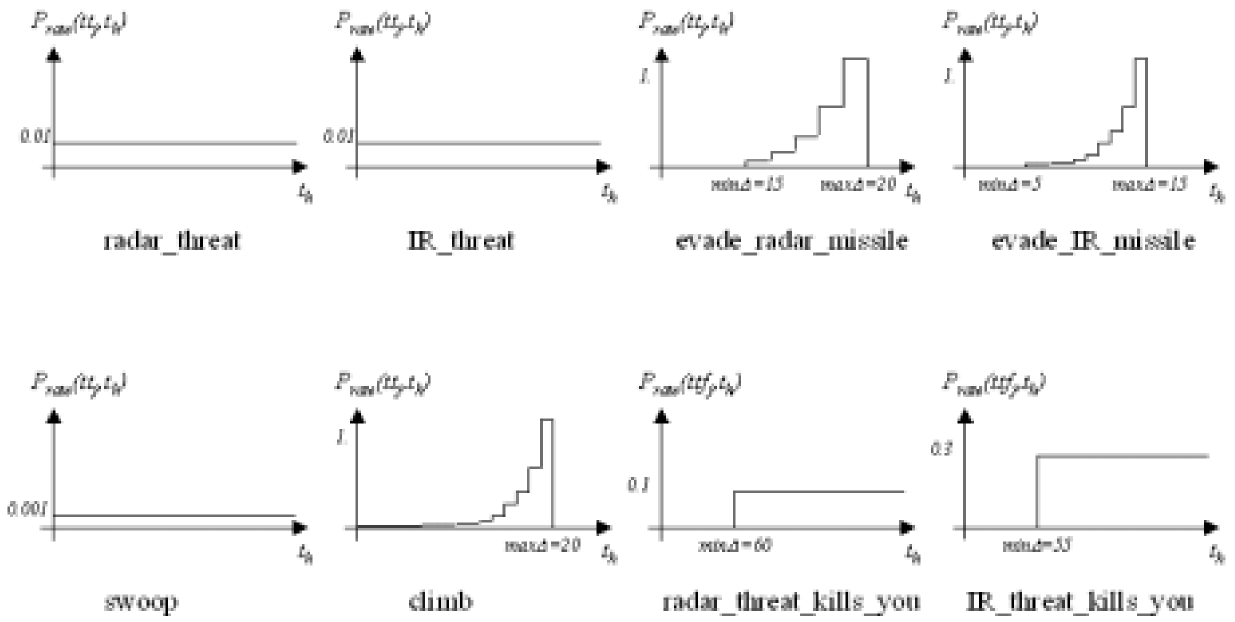


Figure 12. The PSSP probability rate functions for the demonstration domain.

operator. These probability rate functions were defined to set up the key question: if you can only guarantee reaction to one type of missile, which should it be?

One simple approach to trying to answer this question is to look at the probabilities associated with the TTFs and ignore the least likely. In SA-CIRCA, this would correspond to having the AMP not even inform the CSM of some TTFs. There are arguments for and against this approach. The story we stress in this case is that simply looking at the TTFs (the probability of them occurring in a state in which they are enabled) in the knowledge base would lead us to believe that IR missiles are more likely to lead to failure when they are used. An approach that does not model the state space probabilities would thus decide to guarantee against IR missiles instead of radar missiles. This leads to the world model and controller design shown in Figure 13. In this case, only IR missiles are handled because responses could not be scheduled for everything. Of course, if radar-guided missiles are encountered, the plane is frequently destroyed, because the CSM has not even considered their existence.

It turns out, however, that in this domain IR missiles are not actually encountered that often, because the plane is much less likely to be at low altitude. When the PSSP is given the full set of TTFs and generates the state space, it discovers that the likelihood of failing is actually greater for radar missiles: even though they are less likely to lead to failure in a state in which they are enabled than IR missiles are in a state where *they* are enabled, the probability of being in a state with radar missiles enabled is much higher. The resulting controller and world model (state space) are shown in Figure 14.

Qualitatively, then, we might say that, in a particular mission, the probability of

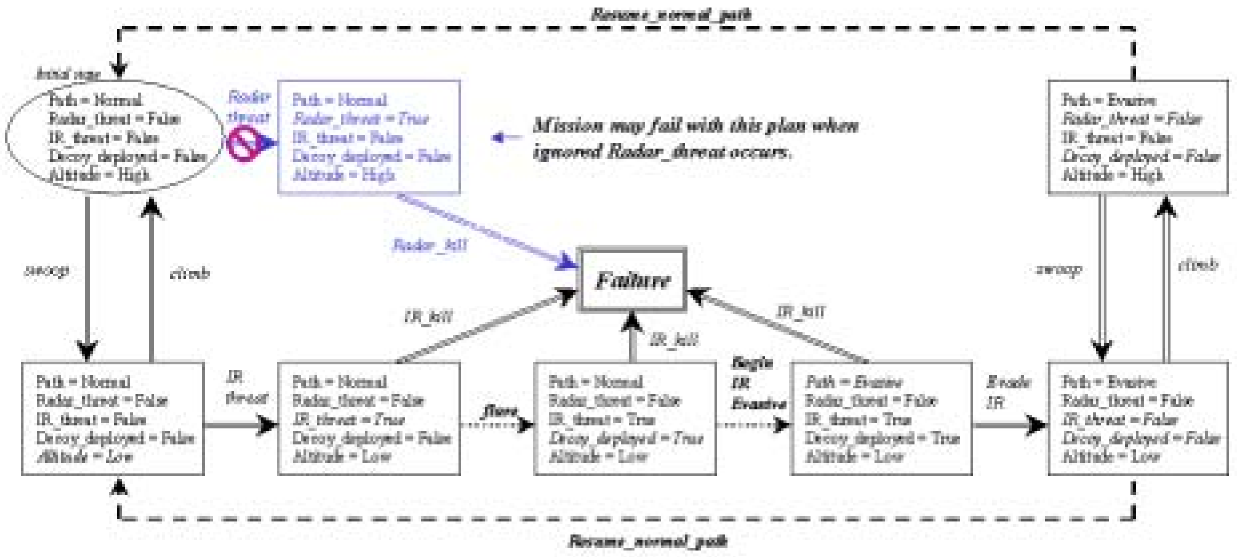


Figure 13. The PSSP state space when only IR missile threats are handled.

encountering no missiles is relatively low, and the probability of encountering only IR missiles is medium. The probability of encountering only radar missiles is high. The probability of encountering both kinds of missiles is low-to-medium. Using the probabilities, the PSSP determines that the no-missile/radar-missile combination is the more likely. So in the above, the plan formed handles the no-missile/IR-missile cases. Using the probability reasoning, the chances of survival are increased.

However, the aircraft may still encounter both IR and radar missiles, and now it is not prepared for the IR type. So we actually allow the PSSP to build plans that explicitly handle regions of the state space that were pruned for earlier controller designs, and cache these new controllers as contingency plans. Figure 15 shows the resulting world model when the PSSP builds the same nominal plan as above, but also builds a detection TAP for recognizing the unhandled (IR missile) case, and pre-builds a contingency plan to handle this case. Note that the contingency plan itself "deadends" in a state where the IR-missile threat has been defeated. The contingency plan has its own detection TAP for reaching a deadend state, which in turn triggers the dispatcher to retrieve a suitable plan - which in this case is simply going back to the nominal plan.

This series of tests illustrates the increasing robustness in the system thanks to modeling state probabilities and detecting/reacting-to unhandled states.

7. Conclusions

The SA-CIRCA research has resulted in the development of two novel planning systems (controller synthesis algorithms). The first combines AI search techniques and heuristics with formal model-checking methods to produce guaranteed, verified real-time control plans with useful formal properties. The second system uses probabilities to optimally

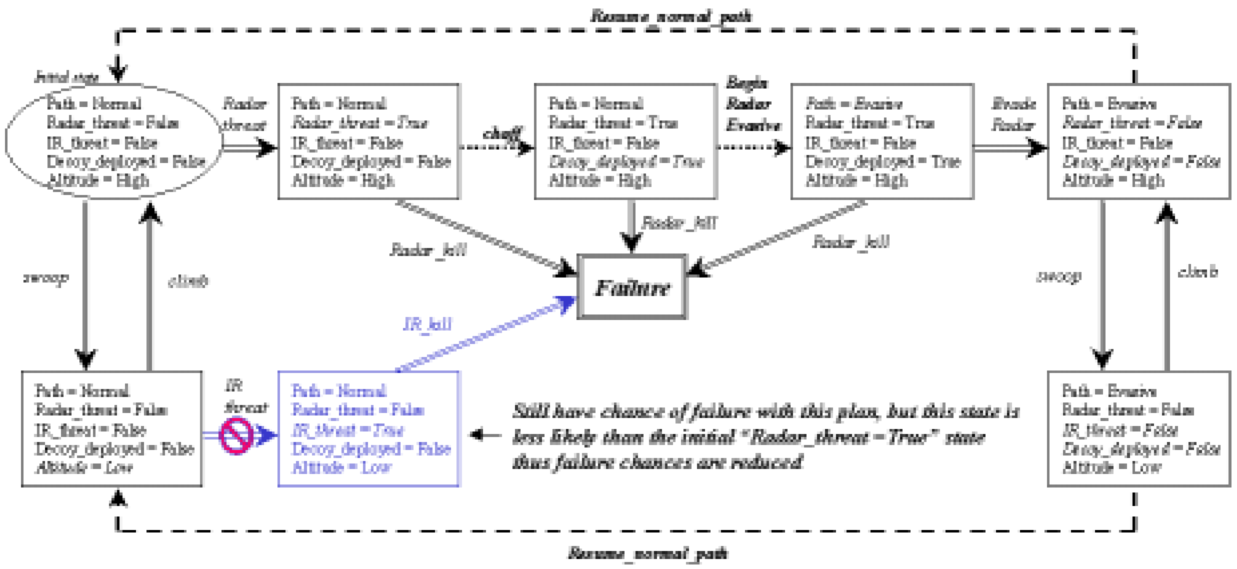


Figure 14. The PSSP state space when only radar missile threats are handled.

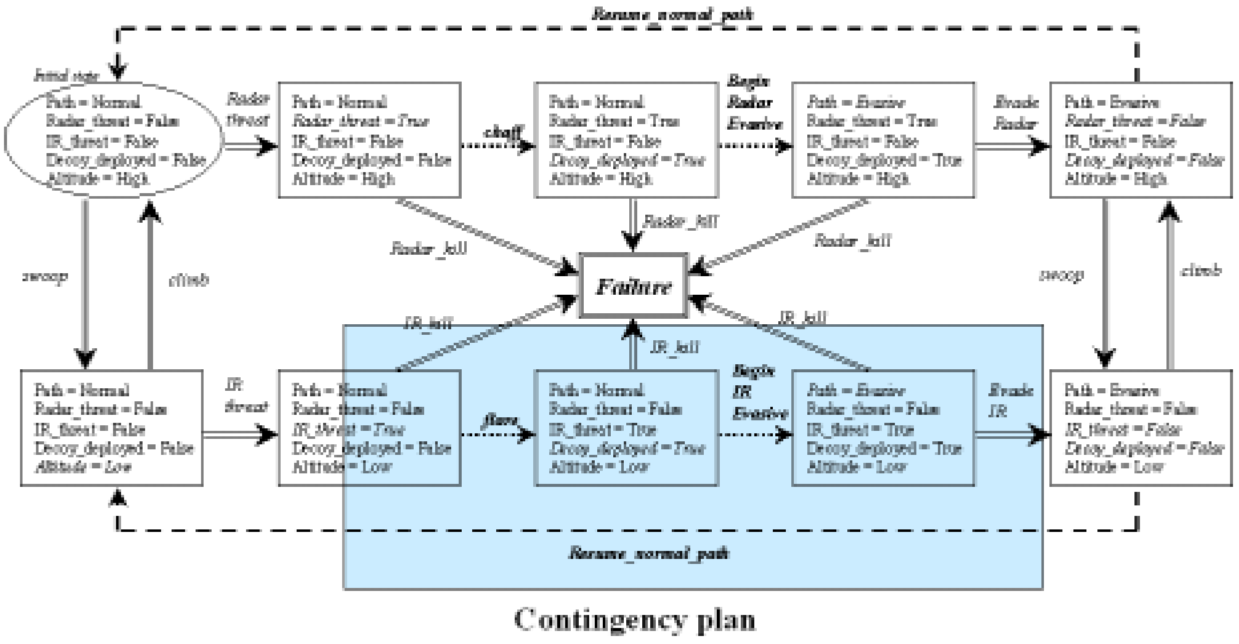


Figure 15. The PSSP state space when radar missile threats are handled in one plan, and a contingency plan is built for IR missile threats.

allocate planning effort to most-probable threats, and develop contingency plans to handle secondary threats that are less likely. These planning techniques will form key elements of future autonomous, self-evaluating, self-adaptive control systems that operate robustly in mission-critical environments.

References

[1] R. Alur, “Timed Automata,” in *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*, 1998.

[2] E. Asarin, O. Maler, and A. Pneuli, “Symbolic controller synthesis for discrete and timed systems,” in *Proceedings of Hybrid Systems II*, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, Springer Verlag, 1995.

[3] C. Boutilier, T. Dean, and S. Hanks, “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, July 1999.

[4] D. L. Dill, “Timing Assumptions and Verification of Finite-State Concurrent Systems,” in *Automatic Verification Methods for Finite State Systems*, J. Sifakis, editor, pp. 197–212, Springer Verlag, Berlin, June 1989. Proceedings of the International Workshop.

[5] M. Fröhlich and M. Werner, “Demonstration of the interactive Graph Visualization System daVinci,” in *Proceedings of DIMACS Workshop on Graph Drawing '94*, R. Tamassia and I. Tollis, editors. Springer Verlag, 1995.

[6] F. Kabanza, “On the Synthesis of Situation Control Rules under Exogenous Events,” in *Theories of Action, Planning, and Robot Control: Bridging the Gap*, C. Baral, editor, number WS-96-07, pp. 86–94. AAAI Press, 1996.

[7] F. Kabanza, M. Barbeau, and R. St.-Denis, “Planning Control Rules for Reactive Agents,” *Artificial Intelligence*, vol. 95, no. 1, pp. 67–113, August 1997.

[8] H. Li, E. Atkins, E. Durfee, and K. Shin, “Resource Allocation for a Limited Real-Time Agent Using a Temporal Probabilistic World Model,” forthcoming, December 1999.

[9] O. Maler, A. Pneuli, and J. Sifakis, “On the synthesis of Discrete Controllers for Timed Systems,” in *STACS 95: Theoretical Aspects of Computer Science*, E. W. Mayr and C. Puech, editors, pp. 229–242, Springer Verlag, 1995.

[10] D. J. Musliner, *CIRCA: The Cooperative Intelligent Real-Time Control Architecture*, PhD thesis, University of Michigan, Ann Arbor, 1993. Available as University of Maryland Computer Science Technical Report CS-TR-3157.

[11] D. J. Musliner, E. H. Durfee, and K. G. Shin, “World Modeling for the Dynamic Construction of Real-Time Control Plans,” *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, March 1995.

Appendix A.

CIRCA-II and the Probabilistic State Space Planner

Resource Allocation for a Limited Real-Time Agent Using a Temporal Probabilistic World Model

Haksun Li, Ella Atkins*, Edmund Durfee, Kang Shin
EECS Dept., University of Michigan, Ann Arbor MI
***Aerospace Engineering Dept., Univ. of Maryland, College Park MD**
{haksunli, durfee, kgshin} @umich.edu, atkins@eng.umd.edu

Abstract

In a dynamic, unpredictable world, a resource-limited agent cannot be ready to recognize and respond fast enough to every possible situation, and thus should prepare for situations that are more likely to occur, and that are more severe. The likelihood of encountering a particular world state, however, is dependent not only on the choices of what (re)actions the agent periodically considers, but also on how frequently it considers them. We have developed and tested a framework for modeling the dynamics of time-dependent event probabilities and an algorithm for computing state probabilities, so that our resource-limited agent can devote its resources to the most probable eventualities over the less probable ones.

1. Introduction

An agent with limited perceptual, computational, and actuator resources must carefully allocate these resources to do the best job it can in monitoring aspects of the world state and acting appropriately to achieve its goals (and maintain its safety) in a complex, dynamic world. If the resource-limited agent allocates more of its resources (or some of its resources more frequently) to monitoring for some states (or state features), then it will be less capable of tracking others successfully. Designing an effective schedule of monitoring for and responding to activities with the available resources, therefore, is a complex optimization problem.

Although some clever techniques, such as indexing, hashing, or grouping states together into action classes may alleviate the problem of allocating limited resources to some extent (Musliner, et al. 1993), the fundamental problem is that monitoring for and responding to all conceivable contingencies by a resource-limited agent in a complex and dynamic domain is not realistic. It is therefore reasonable to have the agent prioritize the use of its resources to prepare for those situations that are most likely to occur.

Determining the likelihood of encountering a particular world state, however, can be challenging because the likelihood is dependent not only on the choices of what (re)actions the agent should perform, but also by its choices of how frequently it checks whether to perform them. By definition, a dynamic environment is one in which the state can change via events outside of the agent's control. Thus, the sooner an agent detects and responds to a situation, the less opportunity there is for the environmental dynamics to intervene. In the context of events that can lead to catastrophe, this simply means that the agent should (re)act fast enough to preempt[1] such catastrophic events.

In this paper, we present a framework for modeling the dynamics of time-dependent event probabilities and an algorithm for computing state probabilities. We thus can prioritize the states of the world by their state probabilities, so that the system can devote its resources to the most probable eventualities over the less probable ones.

To ground our discussion, we describe our time-dependent state-transition probability model in the context of CIRCA-II (Section 2). In Section 3, we describe how we model state transitions with a discrete model, and contrast this with a continuous time model in Section 4. We detail the state probability computation in Section 5 and contrast its performance to previous versions of CIRCA. In section 6, we discuss how a chain of state transitions complicates the problem and how we handle them. Section 7 describes our stochastic simulation to verify the correctness of the theory in practice. Section 8 concludes our works and suggests directions for development.

2. Background

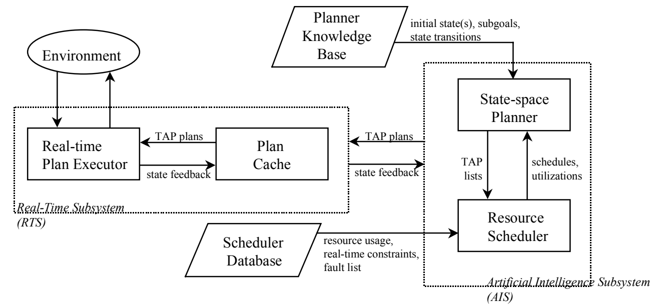
As AI systems are applied in increasingly dynamic and complex domains, it becomes more difficult to make assurances about how well they will perform. Techniques have been under development to support the construction of probabilistic plans (Kushmerick, Hanks, and Weld 1994) for applications where the actions taken by an agent can have uncertain outcomes. Planners based on Markov Decision Processes (MDPs) incorporate representations of uncertainty about how actions and events may move the world among states. MDP planners generate plans (or policies) about what action (if any) an agent should perform for each observable state so as to maximize the agent's expected performance. MDP planning (Dean, Kaelbling, Kirman, and Nicholson 1993) is most straightforward under assumptions such as: that states are *fully observable* (the agent executing the plan can know exactly what state it is in); that event models are *implicit* (the uncertainties about how events beyond the agent's

[1] Preemption thus means that the agent's action occurs before the undesirable event, thus forestalling that event.

control affect the states it can reach via an action it takes are folded into the state transition probabilities associated with the action); and that a transition probability is *stationary* (dependent only upon the state and not upon timing associated with the state). In the terminology of (Boutilier, Dean, and Hanks 1998), these assumptions hold for a stationary, Fully Observable MDP with implicit event models.

Our efforts in developing AI-based agents for real-time applications have forced us to relax some of these assumptions in ways different from typical relaxations (such as for Partially Observable MDPs). In a real-time application, not surprisingly, a core concern is "how long" something will take. Making real-time guarantees means that the agent must take some actions before catastrophic events could occur. That is, between the time when a state arises that heralds an impending event to be avoided (e.g., a slow-moving car enters your lane ahead of you), and the earliest time that that event (e.g., collision) could occur after the heralding state is entered, a real-time agent must be assured of taking an action (e.g., swerving) that averts the undesirable event. Because an agent is possibly moving non-deterministically among many states, it must be prepared to recognize and react fast enough to a number of heralding states. Given limited sensory and processing resources, the set of "recognition-reactions" must be carefully selected and scheduled to ensure each is checked frequently enough to preempt failure.

The Cooperative Intelligent Real-time Control Architecture (CIRCA-II) is an example of a system that has been developed for selecting, scheduling, and executing recognition-reactions in a resource-limited system. CIRCA-II extends the original CIRCA architecture (Musliner et al, 1993, 1995) to use probabilities for making resource allocation decisions and to cache plans for foreseeable but low-probability states. The CIRCA-II architecture is shown in Figure 2-1. Here, we concentrate only on the relevant CIRCA-II features to explain how we compute state probabilities in a real-time environment. More complete treatments of the overall CIRCA-II architecture are presented elsewhere (Atkins 1999).



CIRCA -- The Cooperative Intelligent Real-time Control Architecture.

Figure 2-1

There are two main subsystems in CIRCA-II, the Artificial Intelligence Subsystem (AIS) and the Real-Time Subsystem (RTS). Inside the AIS are the probabilistic planner and the scheduler. The planner has a representation of world states and transitions among them in terms of a state space diagram. If there is any transition leading to a failure state (failure transition) from a state, the planner will plan a guaranteed action such that the failure transition is preempted. Otherwise, it may plan a 'if-time' action, which is only executed should resources allow, to get the system closer to the goal state. A design goal of CIRCA-II's State-Space Planner is that it uses a compact representation of the information needed to generate real-time temporal control plans. Specifically, CIRCA-II does not generate more than one state for any state in the state diagram even if it can be reached via multiple paths from the initial states. Therefore, the state diagram is not necessarily a tree-structured diagram — cyclic paths may be introduced into it. In other words, we do not keep track of the history, and we plan a single action for a group of situations with the same state features (which is represented only by one state in the state diagram), regardless of how that state is reached. The plan developed is thus a recognition-reaction plan. It is precisely this characteristic that allows CIRCA-II to formulate a cyclic schedule in which actions are planned for states (no matter when those states are encountered) so that undesirable transitions cannot occur.

Furthermore, because event probabilities depend on which actions are scheduled and how frequently, implicit event models become untenable because of the large space of potential actions and each possible delay for each action. CIRCA-II therefore explicitly models events. As pointed out by (Boutilier, Dean, and Hanks 1998), specifying a transition function for an action and one or more exogenous events is generally not easy. In what follows, we describe and evaluate our strategy for doing this. Therefore, CIRCA-II's planning could be viewed as a means for formulating an approximately optimal plan for a (partially) non-stationary, FOMDP with explicit event models.

An action planned by the planner together with the test part for state recognition is called a Test-Action-Pair (TAP). The set of TAPs, generated by the planner, is then passed onto the scheduler to find a feasible schedule for all the TAPs. The RTS is a predictable, reactive system that repeatedly loops over the TAP schedule, executing each TAP's test expression (which encapsulates all the relevant sensing actions) and executing its action if the test returns true.

Since the scheduler is working with a RTS with limited resources, it could be the case that not all of the requested TAPs can be scheduled. When this occurs, the planning system could identify and try out alternative TAP combinations in the hope of finding one that works. Often, however, the RTS simply may be incapable of guaranteeing any combination of TAPs that completely assures that all critical real-time responses are met. In such cases, it is up to the planner to find a schedulable set of TAPs that assures that as many of the most critical responses are met as possible.

In this paper, we will concentrate on the planner, which employs the new probabilistic temporal transition framework, and plans selectively based on state probabilities given the resource constraints of the RTS. Here we summarize the algorithm of the planner, which is the setting for the discussion in the later sections. The planner, during each planning cycle, selects the most probable state and "expands" it by applying all enabled event transitions (whose conditions are met in the state) and computing their time-dependent transition probabilities. Depending on the possible consequences of these events, the planner may select an action for the state, and the action's resulting state will be added to the set of possible successor states of the action. The probabilities of each of these states are computed, and the planning cycle loops. If one of these states has already been generated in an earlier planning cycle, its probability (and the probabilities of its successors, if any have yet been generated) must be updated. The planner continues in this cyclic manner until all reachable states have been investigated, or until only the states whose probabilities fall below some threshold parameter remain. If one of the low probability "unplanned" states is actually reached, CIRCA-II has mechanisms to detect it and replan or retrieve a contingency plan, as are discussed elsewhere (Atkins 1999).

3. Temporal Transitions

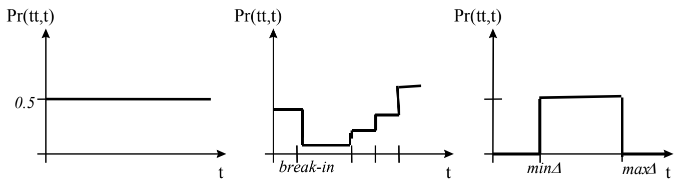
The CIRCA-II world model is constructed from a set of state features and state transitions included as part of the planner knowledge base. A state in the world model is created dynamically by applying a transition to its parent state during planning. There are two types of transitions. Action transitions are explicitly controlled by the plan executor in the RTS, and thus only occur when selected during planning. Events outside the system's control are modeled as temporal transitions, either "innocuous" temporal transitions (labeled *tt*) or "malicious" temporal transitions to system failure (labeled *ttf*).

CIRCA-II's primary objective is to avoid system failure. It considers goal achievement only when the safety requirements are met. When expanding a state, CIRCA-II bases its action selection primarily on failure avoidance when a *ttf* is present, only considering proximity to the goal when more than one action is capable of avoiding failure. Alternatively, if no *ttf* is present, CIRCA-II selects the best action, called a 'if-time' action, to achieve the system goals, as described in (Musliner 1993). If no action is required for failure avoidance or needed for goal achievement, CIRCA-II selects no action.

As mentioned in the previous sections, we have to model the likelihood of a transition as a function of time to capture the dynamic nature of the world. We will detail in what follows our framework for modeling temporal transitions.

For a temporal transition tt_j, the user specifies what is called a "probability rate" function $P_{rate}(tt_j,t_h)$ over a time interval $[t_h,t_{h+1})$ (e.g., seconds), given that transition tt_j has been continuously active for h time steps. For example, if a fair coin is flipped once per second, the "probability rate" function for the transition from "heads" to "tails" has a constant value of 0.5 (50%) over each second, regardless of how much time has passed, given that the state is "heads" after the flips so far. Figure 3-1a shows the probability rate function for this coin flip example.

Many realistic state transitions can be easily defined in terms of "probability rate". Figure 3-1b describes when an engine is first put into service, its "failure rate" decreases during a break-in period. Afterwards, the rate is very small during the normal operation period. When the engine nears the end of its life (e.g., 2000 hours for a small propeller-driven engine), the failure rate increases until the engine is considered "unsafe" and must be retired.



a) Heads-to-tails *tt*. b) Engine failure *tt*. c) Minimum delay *ttf*.

Figure 3-1: Temporal transition probability rate functions.

Figure 3-1c shows a transition probability rate function for a temporal transition with a minimum delay (*minΔ*) before it is possible and a maximum delay (*maxΔ*) after which the transition cannot occur. CIRCA-II must select an action to preempt this *ttf* provided such an action can be scheduled before time *minΔ*.

Action transitions, like temporal transitions, are also described by probability rate functions, such that the conditional probabilities of the transitions from a state can be computed. Equation 3-1 (where i stands for the i-th time interval $[t_i, t_{i+1})$) shows the probability rate function we adopt for guaranteed actions. The important point to note is that the action has an associated maximum period (*maxp*) in which it must execute to preempt its *ttf(s)*.

$$Pr(ac,t_i) = \begin{cases} \dfrac{1}{(\text{maxp} - i)} & (0 \leq \text{i} < \text{maxp}) \\ 0 & (\text{i} \geq \text{maxp}) \end{cases} \qquad (3\text{-}1)$$

For 'if-time' (best-effort) actions, which are executed in slack time slots produced when guaranteed actions do not require their worst-case execution time (Musliner 1993), the probability rate function is for now a constant value that is only a heuristic guess and can be assigned by the user. In future work, we plan to incorporate a function similar to that for guaranteed actions to more accurately estimate the timing information associated with 'if-time' action transitions.

The probability rate function captures the probability of an event occurring independently of other events over any time interval. To model the dependency among a set of *tt*s that match a state, we can approximate the conditional probability rate function $P_{cond}(trans_j, t_h, s_k)$ for transition $trans_j$ from state s_k during time interval $[t_h,t_{h+1})$ heuristically as given by Equation 3-2.

$$P_{cond}(trans_j, t_h, s_k) \approx \frac{\ln(1 - P_{rate}(trans_j, t_h, s_k))(1 - P_{rate}(none, t_h, s_k))}{\sum_{\forall trans_r \in trans(s_k)} \ln(1 - P_{rate}(trans_r, t_h, s_k))} \qquad (3\text{-}2)$$

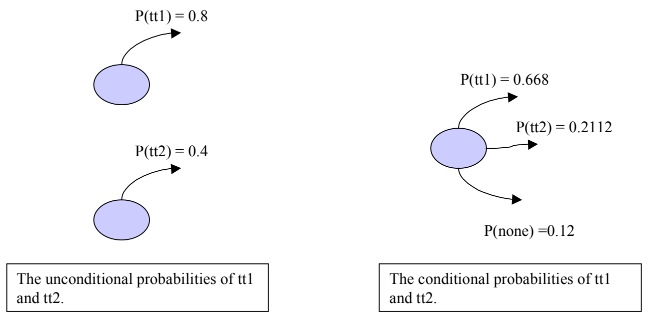
$P_{rate}(trans_j,t_h,s_k)$ is the unconditional probability of transition $trans_j$ from state s_k during time interval $[t_h,t_{h+1})$. In a continuous time model, the probability of two transitions firing simultaneously is vanishingly small, but in our discrete case we are considering time intervals. Hence, there is a non-zero probability that two (or more) transitions could fire over the same time interval. Because we only want to allow one transition to fire (presumably, one would come before the others in the time interval), we have to divide the probability of a combination of transitions among the separate transitions. Rather than dividing it equally (treating each transition in the combination as being equally likely to occur first in the interval), our intuitions (borne out in Section 4) are that more probable transitions jump in front more often.

Thus, Equation 3-2 is essentially saying that the conditional probability in time interval $[t_h,t_{h+1})$ of a transition from state s_k is proportional to the relative logarithm of one minus its likelihood among all other transitions, given that the system is still in state s_k in the time interval. We will explain in Section 4 the significance of this heuristic. Currently, $P_{rate}(\text{none}, t_h, s_k)$ is calculated using 3-3.

$$P_{rate}(none,t_h,s_k) = \prod_{\forall trans_r \in trans(s_k)} (1 - P_{rate}(trans_j, h_h, s_k)) \qquad (3\text{-}3)$$

Equations 3-2 and 3-3 are heuristics to estimate the conditional probabilities from the unconditional probabilities given the compact representation that does not capture conditional dependency information among the temporal transitions. Two assumptions are made in 3-2 and 3-3 to model the conditional dependencies. First, we are assuming that the probability of nothing happening is calculated as if all temporal transitions were independent. Second, we are assuming also that the conditional probabilities of the transitions in each time step are proportional to their unconditional probabilities.

As a simple example, let tt_1 and tt_2 be two temporal transitions which have unconditional probabilities of 0.8 and 0.4 respectively at each time step. From our equations, the conditional probabilities in any time interval are as shown in Figure 3-3. Equation 3-5 tells us that the transition probabilities are 0.759 for tt_1 and 0.241 for tt_2.



P(tt1) = 0.8
P(tt2) = 0.4
The unconditional probabilities of tt1 and tt2.
P(tt1) = 0.668
P(tt2) = 0.2112
P(none) =0.12
The conditional probabilities of tt1 and tt2.

Figure 3-3

In general, the problem of calculating conditional probabilities from only unconditional probabilities is difficult. Our heuristic, based on the two biases we have chosen, is only one of the many ways to do the estimation. In Section 4, we will consider this example using continuous-time models to justify our heuristic.

To compute the state probability, CIRCA-II calculates the transition probability for $trans_j$ from state s_k to another using the (overall) cumulative probability $P_{cum}(trans_j, s_k)$. This describes the relative probability of leaving state s_k via $trans_j$, and is equal to the sum of $P_{cond}(trans_j, t_h, s_k) * P(s_k, t_h)$ over time. $P(s_k, t_h)$ is the probability of the system still being in state s_k in time interval t_h, and it is recursively calculated using 3-4, where $P(s_k, 0) = 1.0$.

$$P(s_k, t_h) = P(s_k, t_{h-1}) P_{rate}(none, t_{h-1}, s_k) \quad (3\text{-}4)$$

Equation 3-5 computes the transition probability from the initial time interval, (t_0, t_1), up to the "converged" time interval (t_n, t_{n+1}) when either the likelihood of being in state s_k is below a preset threshold ε (i.e., $P(s_k,t_n) < \varepsilon$) or after which all the transition probability rates are negligible ($P_{cond}(trans_j, t_i, s_k) < \varepsilon$), where i > n.

(3-5)

$$P_{cum}(trans_j, s_k) = \sum_{h=0}^{\infty} P_{cond}(trans_j, t_h, s_k) P(s_k, t_h)$$

4. Heuristic Justification

Equation 3-2 describes how we approximate the relative likelihood of all the temporal transitions in a time interval. The heuristic divides the probability of transitioning out from a state in a particular time interval among the alternative transitions in proportion to the values of their probability rate functions in that time interval. This ensures that the probability of following multiple transitions at the same time interval is zero, because (unless it is modeled as an explicit transition) the probability of two transitions occurring at exactly the same time point is zero.

We can compare the results from our heuristic in the discrete time interval case with the results one could get using a continuous time model. Our analysis is two-folded. We will first show the case in which there are only temporal transitions in a state. Then, we will consider the case in which a state has a guaranteed action.

To simplify our analysis, we begin by assuming the simplest case in which all probability rate functions for temporal transitions are constant, as in the example in Figure 3-3. If there are only temporal transitions in a state, we can pose the question mathematically: let $tt_1, tt_2, \ldots, tt_n$ be n different independent temporal transitions which have constant rates, $r_1, r_2, \ldots r_n$. T_i is the time that transition tt_i fires. $T = \min\{T_1, T_2, \ldots T_n\}$. What is the probability that $P\{T_i = T\}$?

When a temporal transition has a constant rate, the probability of it firing in any time interval $[t, t+\Delta t)$ is the same as any other time interval. Another way of saying this is that the probability of a transition firing after time t+s, given that time s has elapsed is the same as the probability of the transition firing after time t:

$$P(T_i \geq s + t \mid T_i \geq s) = P(T_i \geq t)$$

The only real-valued functions satisfying $f(s + t) = f(s)f(t)$ are of the form $f(t) = e^{-\lambda t}$. Hence, the distribution of T_i must be an exponential distribution with parameter λ. To summarize, we have:

- The probability of tt_i firing after time t is $e^{-\lambda t}$.
- The cumulative probability function of tt_i, i.e., the probability of tt_i firing before time t, is $1 - e^{-\lambda t}$.
- The probability density function of tt_i is $\lambda e^{-\lambda t}$.

For independent transitions:

$$\begin{aligned} P(T_i > t) &= P(T_1 > t, T_2 > t, \ldots, T_n > t) \\ &= P(T_1 > t) P(T_2 > t) \ldots P(T_n > t) \\ &= e^{-\lambda_1 t} e^{-\lambda_2 t} \ldots e^{-\lambda_n t} \\ &= e^{-(\lambda_1 + \lambda_2 + \ldots \lambda_n) t} \end{aligned}$$

To calculate the probability that tt_i fires first:

$$
\begin{aligned}
P(T_i = T) &= \int_0^\infty P(T_1 > t, \ldots, T_{i-1} > t, T_{i+1} > t, \ldots, T_n > t)\, dP\{T_i = t\} \\
&= \int_0^\infty e^{-\lambda_1 t} \ldots e^{-\lambda_{i-1} t} e^{-\lambda_{i+1} t} \ldots e^{-\lambda_n t} \lambda_i e^{-\lambda_i t}\, dt \\
&= \int_0^\infty e^{-(\lambda_1 + \ldots + \lambda_{i-1} + \lambda_{i+1} + \ldots \lambda_n)t} \lambda_i e^{-\lambda_i t}\, dt \\
&= \frac{\lambda_i}{\lambda_1 + \ldots + \lambda_n} \qquad (4\text{-}1)
\end{aligned}
$$

Therefore, the relative likelihood of *tt*$_i$ is proportional to λ_i among those of all temporal transitions.

We are now ready to connect λ to rate r in the discrete case. Let us assume that in the discrete case, the size of a time interval is 1. So, the probability of *tt*$_i$ firing after t, i.e, t time intervals, is $(1 - r)^t$. Equating this with the continuous time counterpart above, we get:

$$
\begin{aligned}
(1 - r)^t &= e^{-\lambda t} \\
1 - r &= e^{-\lambda} \\
r &= 1 - e^{-\lambda} \\
\lambda &= -\ln(1 - r)
\end{aligned}
$$

Therefore, it is possible to convert from the discrete rate, r, to the continuous rate, λ, and vice versa. Their relationships are:

$$
\begin{aligned}
r &= 1 - e^{-\lambda} \\
\lambda &= -\ln(1 - r)
\end{aligned}
$$

If we substitute *r* into the (4-1), we get

$$
\begin{aligned}
P(T_i = T) &= \frac{\lambda_i}{\lambda_1 + \ldots + \lambda_n} \\
&= \frac{-\ln(1 - r_i)}{-\ln(1 - r_1) + \ldots + -\ln(1 - r_n)} \\
&= \frac{\ln(1 - r_i)}{\ln(1 - r_1) + \ldots + \ln(1 - r_n)}
\end{aligned}
$$

This is essentially the logarithm proportionality heuristic in Equation 3-2, and using this formulation we get the same transition probabilities as in Figure 3-2. λ_{tt1} is 1.609 and λ_{tt2} is 0.511. The transition probabilities are therefore 0.759 for *tt*$_1$ and 0.241 for *tt*$_2$. This agrees with what is calculated by Equation 3-5.

Now, let us consider a case that involves a non-constant function. Often in CIRCA's state space diagram, a state may have a guaranteed action if a *ttf* has to be preempted. To go through the same analysis above, we will have to have the probability density function for a guaranteed action, *ac*, which has a period P. According to our definition of a guaranteed action transition, and as in Section 3 using the approximation[2] that the action is equally likely to fire at any time within P, we have the following:

- The cumulative probability function of *ac*, i.e., the probability of *ac* firing before time t is $\frac{t}{P}$, if $0 \le t \le P$
  0 , if $t > P$
- Taking the derivate of the cumulative probability function, the probability density function of *ac* is
  $\frac{1}{P}$, if $0 \le t \le P$
  0 , if $t > P$
- The probability of *ac* firing after time t is $\frac{P - t}{P}$, if $0 \le t \le P$
  0 , if $t > P$

We can derive the probability rate function from these equations. The probability of an *ac* firing in the time interval (t_1, t_2) given that it has not fired before t_1 is

$$
\begin{aligned}
P(t_1 \le T \le t_2 \mid t_1 < T) &= \frac{P(t_1 \le T \le t_2)}{P(t_1 \le T)} \\
&= \frac{t_2 - t_1}{P - t_1}
\end{aligned}
$$

As t_2 moves toward P, keeping the time interval (difference between t_1 and t_2) unchanged, the probability rate goes to 1 as we would expect.

As in the analysis above, to calculate the probability that *tt*$_i$ fires first:

$$
\begin{aligned}
P(T_i = T) &= \int_0^P P(T_1 > t, \ldots, T_{i-1} > t, T_{i+1} > t, \ldots, T_n > t, ac > t)\, dP\{T_i = t\} \\
&= \int_0^P \left(e^{-\lambda_1 t} \ldots e^{-\lambda_{i-1} t} e^{-\lambda_{i+1} t} \ldots e^{-\lambda_n t}\right)\left(\frac{P - t}{P}\right) \lambda_i e^{-\lambda_i t}\, dt \\
&= \int_0^P e^{-(\lambda_1 + \ldots + \lambda_{i-1} + \lambda_{i+1} + \ldots \lambda_n)t} \lambda_i e^{-\lambda_i t} \left(\frac{P - t}{P}\right) dt \\
&= \lambda_i \left[\int_0^P e^{-\Lambda t}\, dt - \frac{1}{P}\int_0^P t e^{-\Lambda t}\, dt\right] \\
&= \frac{\lambda_i}{\Lambda}\left(1 - e^{-\Lambda P}\right) + \frac{\lambda_i}{\Lambda} e^{-P\Lambda} - \frac{\lambda_i}{P\Lambda^2}\left(1 - e^{-\Lambda P}\right)
\end{aligned}
$$

, where $\Lambda = \sum_{i=1}^{n} \lambda_i$

This answer is much more complicated than the logarithm heuristic we had before because of the added consideration of a guaranteed action. In fact, when $P \to \infty$ (an infinite period means that the probability of the action firing in a finite time approaches zero), this expression becomes $\frac{\lambda_i}{\Lambda}$, which is exactly what we had before.

We will also want to find the probability of the action, *ac*, being the first transition to fire.

[2] This is only an approximation because it ignores lag time introduced by sensing, and also the "jump" a transition might get if it is part of a dependent chain (see Section 5).

$$
\begin{aligned}
P(T_i = T) &= \int_0^P P(T_1 \geq t, \ldots, T_n \geq t) dP\{T_{ac} = t\} \\
&= \int_0^P e^{-\lambda_1 t} \ldots e^{-\lambda_n t} \frac{1}{P} dt \\
&= \int_0^P e^{-(\lambda_1 + \ldots + \lambda_n)t} \frac{1}{P} dt \\
&= \frac{1 - e^{-P\Lambda}}{P\Lambda}
\end{aligned}
$$

$$, \text{where } \Lambda = \sum_{i=1}^{n} \lambda_i$$

We have done a sanity check to verify that the sum of all the probabilities of any transitions firing first indeed equals 1. (This can be worked out by the reader.)

To verify how well our heuristic approximation in the discrete case works when a state also has a guaranteed action, we consider the following example: a state has tt_1 of constant probability rate 0.1, tt_2 of 0.6, and an action with period 5. In general, the more finely we discretize the time intervals, the better results we get. Here is a summary of our calculations.

For the action transition:

number of time intervals (discreti-zation level)	time each time interval represents	discretized calculation value: action transition probability	true value as computed in continuous time domain	error rate (per-centage)
5	1 sec	0.200162756	0.194578	2.870137
10	0.5 sec	0.195908691	0.194578	0.683835
100	0.05 sec	0.194591233	0.194578	0.006751

For tt1 transition:

number of time intervals (discreti-zation level)	time each time interval represents	discretized calculation value: tt1 transition probability	true value as computed in continuous time domain	error rate (per-centage)
5	1 sec	0.0824	0.083059	0.79281
10	0.5 sec	0.0829	0.083059	0.19083
100	0.05 sec	0.08306	0.083059	0.001806

For tt2 transition:

number of time intervals (discreti-zation level)	time each time interval represents	simulation value: tt2 transition probability	true value as computed in continuous time domain	error rate (per-centage)
5	1 sec	0.71735	0.722361	0.69366
10	0.5 sec	0.721167	0.722361	0.16525
100	0.05 sec	0.722349	0.722361	0.00162

Again, as suggested by the data, the more we discretize the time intervals, the more closely our discrete approximation matches the continuous time model even when some transitions are not constant rate functions. Future work will examine the limits of our discrete approximation, such as how well it works with more arbitrary functions. Techniques for solving a general problem with arbitrary rate functions in a continuous time domain can be complicated and costly. On the contrary, our discrete approximation does not increase much in complexity even if the probability rate functions are arbitrary. If we can prove what our results above suggest --- that the discrete approximation can give arbitrarily accurate results by discretizing more finely regardless of the shapes of the transition functions, then our approximation provides a simple, yet effective and powerful method to compute transition probabilities from domain-dependent transition functions. As will be discussed in the later sections, from these transition probabilities, we can compute the state probabilities reasonably well.

5. State Probabilities

With our framework for computing transition probabilities, we can estimate the probability of the system ever entering a particular state in order to decide whether to allocate resources to responding to that state. After computing the transition probabilities, as described in section 3, we can construct a state diagram, in which each node corresponds to exactly one state. The state diagram can be represented by a matrix M_{ij} as in Figure 5-1. M[i][j] is the transition probability from node i to node j (we will use "node" and "state" interchangeably). Note that, given our characterization so far, the transition probabilities satisfy the Markov property. That is, once we have computed the probability of transitioning from one state to another based on the various time-dependent probability rate functions, we can form the state transition diagram. Given the diagram, we are now ready to compute the state probabilities. We present a theorem from (Kemeny and Snell 1960):

The probabilities of going from any nodes in the transient set (a transient node is one which has outgoing transitions, i.e., *tt*s, *ttf*s, and/or actions) to any nodes in the absorbing set (an absorbing node is one which has no outgoing transitions) are given in the matrix:

$$P = NR, \quad where \quad N = (1 - Q)^{-1} \tag{5-1}$$

R is the matrix indicating the probabilities of going from the transient nodes (row indices) to the absorbing nodes (column indices) in one step. Q is the matrix indicating the probabilities of going from the transient nodes (row indices) to other transient nodes (column indices) in one step.

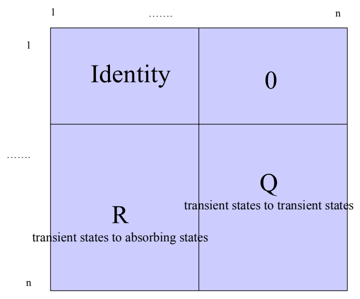

1
....
n
1
....
n
Identity
0
Q
transient states to transient states
R
transient states to absorbing states

Figure 5-1

Both matrices, R and Q, are readily available from the transition matrix M_{ij}. To compute the state probabilities of all absorbing nodes in the state diagram, we only need to apply 5-1 once. However, to compute the probability of a transient node, we make it into an absorbing node by truncating all its outgoing transitions. Since we are only concerned with the probability of ever visiting a node, all paths coming out from the node will not further affect the state probability even if the path subsequently enters into the node again, i.e. a cycle. Unfortunately, N and R will thus be different in each computation of the state probability of each transient node. The cost of computing the state probabilities for transient states will therefore be higher than that for absorbing states because equation 5-1 has to be used for each transient state probability.

Here we outline the algorithm of computing the state probabilities of all nodes in a state diagram. Let G be a state diagram with *n* nodes, *i* initial state nodes, *p* absorbing nodes, and *q* transient nodes. Initial states are assumed transient. Therefore $p + q = n$.

1. Index the nodes of G such that the indices of the absorbing nodes are less than those of the transient nodes. Construct the transition matrix M for G for this particular indexing.
2. Compute R and Q from M. R = M[p+1 … n][1 … p]. Q = M[p+1 … n][p+1 … n].
3. Compute $N = (1 - Q)^{-1}$.
4. Compute P = NR.
5. P is a $q * p$ matrix giving the probability P_{ij} from any transient node *i* to any absorbing node *j*. Therefore, the sum of the probabilities from all the initial state nodes to an absorbing node is the state probability of the absorbing node. At this point, we have computed all the state probabilities for all absorbing nodes.
6. For each transient node T, do:
    i. Truncate all outgoing transitions of T.
    ii. Index the nodes in G such that the indices of the absorbing nodes are less than those of the transient nodes. T has index 1. Construct a transition matrix M for G for this particular indexing.
    iii. Compute R and Q from M. R = M[p+2 … n][1 … p+1]. Q = M[p+2 … n][p+2 … n].
    iv. Compute $N = (1 - Q)^{-1}$.
    v. Compute P = NR.
    vi. P is an $(q - 1)* (p + 1)$ matrix giving the probability P_{iT} from any transient node i to the absorbing node T. Therefore, the sum of the probabilities from all the initial state nodes to T is the state probability of T.

Algorithm 5-1: Computing state probabilities

To see how this algorithm improves CIRCA-II's performance, we will compare the results of computing the state probability by the previous CIRCA-II method without an update mechanism (Atkins 1996), and this new CIRCA-II formulation using the example in Figure 5-2.
This example illustrates the situation of an aircraft trying to maintain its altitude (perhaps in turbulence) and flying to FIX2. State 1 has two transitions, *tt2* and *ac1*, which have constant probability rate functions of 0.9 and 0.5 respectively. The transition probabilities are 0.6429 and 0.3571 respectively as computed by Equation 3-2 and 3-3. The previous CIRCA-II would predict that the likelihood of reaching state 2 and state 4, the destination, are only 0.3571 (Atkins 1996), because it ignored the contribution of *ac2* which was only known to the planner after expanding state 1. However, the probabilities should be 1.0 as computed by the new formulation, algorithm 5-1. As the flight is able to restore a safe altitude at all times, it can fly to its destination with certainty. This is something that the old CIRCA-II would not conclude!

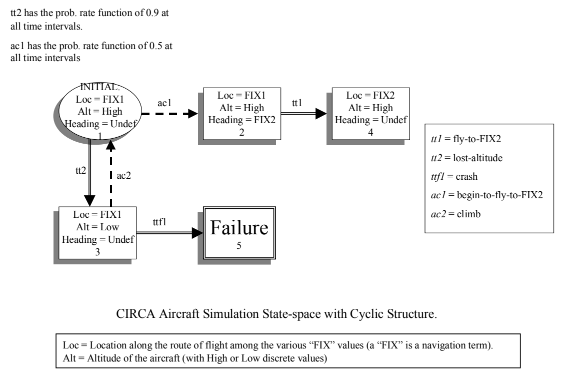

tt2 has the prob. rate function of 0.9 at all time intervals.
ac1 has the prob. rate function of 0.5 at all time intervals
INITIAL: Loc = FIX1 Alt = High Heading = Undef 1
ac1
Loc = FIX1 Alt = High Heading = FIX2 2
tt1
Loc = FIX2 Alt = High Heading = Undef 4
tt2
ac2
Loc = FIX1 Alt = Low Heading = Undef 3
ttf1
Failure 5
tt1 = fly-to-FIX2
tt2 = lost-altitude
ttf1 = crash
ac1 = begin-to-fly-to-FIX2
ac2 = climb
CIRCA Aircraft Simulation State-space with Cyclic Structure.
Loc = Location along the route of flight among the various "FIX" values (a "FIX" is a navigation term).
Alt = Altitude of the aircraft (with High or Low discrete values)

Figure 5-2

Algorithm 5-1 can accurately compute the state probabilities given that the transition probabilities are accurate. We described in Section 3 a means of approximating these probabilities, but as we will see in the next section, sometimes the calculations can be more problematic.

6. Dependent Temporal Transitions

Although a temporal transition can fire in any of the states, the probability rate functions of the same temporal transition may differ across states depending on how they are reached. When a temporal transition is triggered across a sequence of states, its rate function in a later state must consider the time spent in prior states. The time-dependent probability rate functions of the event in each of the states must take into consideration the time spent across multiple states, such as tt_1 in Figure 6-1. We call such temporal transitions dependent temporal transitions (labelled *dtt*) because their probability rate functions depend on their parent states. For example, suppose the probability of engine failure, tt_1, increases with time. Then the probability of tt_1 occurring is higher in FIX2 than in FIX1 because tt_1 is already "enabled" in FIX1 before the flight enters FIX2.

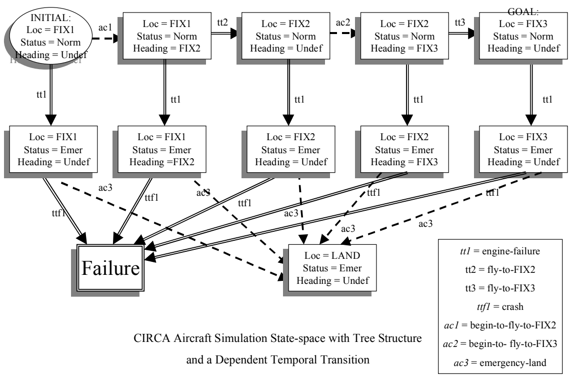

INITIAL: Loc = FIX1 Status = Norm Heading = Undef
ac1
Loc = FIX1 Status = Norm Heading = FIX2
tt2
Loc = FIX2 Status = Norm Heading = Undef
ac2
Loc = FIX2 Status = Norm Heading = FIX3
tt3
GOAL: Loc = FIX3 Status = Norm Heading = Undef
tt1
Loc = FIX1 Status = Emer Heading = Undef
Loc = FIX1 Status = Emer Heading =FIX2
Loc = FIX2 Status = Emer Heading = Undef
Loc = FIX2 Status = Emer Heading = FIX3
Loc = FIX3 Status = Emer Heading = Undef
ttf1
ac3
Failure
Loc = LAND Status = Emer Heading = Undef
tt1 = engine-failure
tt2 = fly-to-FIX2
tt3 = fly-to-FIX3
ttf1 = crash
ac1 = begin-to-fly-to-FIX2
ac2 = begin-to- fly-to-FIX3
ac3 = emergency-land
CIRCA Aircraft Simulation State-space with Tree Structure and a Dependent Temporal Transition

Figure 6-1

Suppose state s_k has a dtt, tt_i. If we knew from which parent s_p the system entered s_k via $trans_j$, and the time t_g it spent in s_p, then we could model the probability rate function of the dtt, tt_i, using Equation 6-1.

$$P_{dtt}\left(tt_i, t_h, s_p, trans_j\right) = P_{rate}(tt_i, t_g + t_h, s_p) \qquad (6\text{-}1)$$

However, this piece of information is not available during planning. In general, there are multiple parents to state s_k, and the system can stay in any of the parent states for a variable amount of time. Each combination of parent state s_p and time spent in s_p, t_g, gives rise to a particular "shifted" probability rate function for the dtt, as described in Equation 6-1. However, we can take the average of these probability rate functions by weighting them by the probabilities of the time t_g spent in s_p, i.e., $P_{cond}(trans_j, t_g, s_p)$. The sum of these weights is simply the cumulative probability of $trans_j$ from s_p to s_k, i.e. $P_{cum}(trans_j, s_p)$.

Equation 6-2 and 6-3 describe how CIRCA-II models a dtt based on the probability rate functions of the dtt in the parent states.

(6-2)

$$P_{dtt}\left(tt_i, t_h, s_p, trans_j\right) = \begin{cases} P_{rate}\left(tt_i, t_h\right) & \text{when} \quad tt_i \notin \left\{trans\left(s_p\right)\right\}, \\ \displaystyle\sum_{t_g=0}^{\infty} \frac{P_{cond}\left(trans_j, t_g, s_p\right) P_{rate}\left(tt_i, t_g + t_h, s_p\right)}{P_{cum}\left(trans_j, s_p\right)} & \text{when} \quad tt_i \in \left\{trans\left(s_p\right)\right\}. \end{cases}$$

$P_{dtt}(tt_i, t_h, s_p, trans_j)$ is the "shifted" probability rate function to reflect the effects of dependent temporal transition (*dtt*), given that the current state s_k is reached via transition $trans_j$ from parent state s_p. If there is no tt_i in s_p, then the probability rate function is "unshifted", i.e. it is simply the unmodified probability rate function. Otherwise, it is the sum of all probability rate functions "shifted" by all possible time delays in s_p weighted by their relative probabilities.

The probability rate function $P_{dtt}(tt_i, t_h, s_k)$ for a dependent temporal transition tt_i from state s_k in the h-th time interval $[t_h, t_{h+1})$, is:

$$P_{dtt}\left(tt_i, t_h, s_k\right) = \left(\frac{1}{P_{initial}\left(s_k\right) + \displaystyle\sum_{\forall(p,j)\ni\left(s_p \xrightarrow{trans_j} s_k\right)} P\left(s_p\right) P_{cum}\left(trans_j, s_k\right)}\right) \times \qquad (6\text{-}3)$$
$$\left(P_{initial}\left(s_k\right) P_{rate}\left(tt_i, t_h\right) + \sum_{\forall(p,j)\ni\left(s_p \xrightarrow{trans_j} s_k\right)} P\left(s_p\right) P_{cum}\left(trans_j, s_k\right) P_{dtt}\left(tt_i, t_h, s_p, trans_j\right)\right)$$

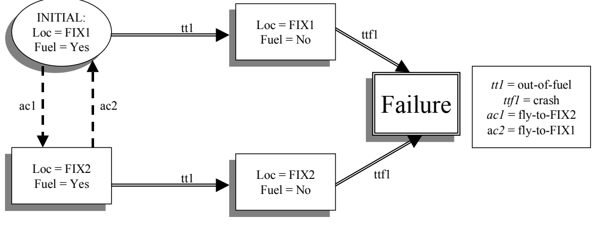
where $P(s)$ is the state probability of state s.

Equation 6-3 is essentially saying that $P_{dtt}(tt, t_h, s_k)$ is a weighted average of all possible "shifted" *dtt*s from all parents, and it is justified in Theorem 6.1.

Theorem 6-1: If the states which have the dependent temporal transitions comprise an acyclic graph, then the probability rate function estimated by Equation 6-3 is the best in terms of minimizing the mean square error at any given time interval.

Proof: In full paper.

Complication arises when we try to apply the formulation to world models with cyclic structures.



CIRCA Aircraft Simulation State-space (Holding Pattern) with a Cycle and a Dependent Temporal Transition for All States in the Cycle

Figure 6-2

As in Figure 6-2, to accurately model the dependent temporal transition tt_1 from the initial state using Equation 6-3, it requires the probability rate function of the other tt_1, which, unfortunately depends in turn on the tt_1 in the initial state. A reasonable heuristic to compute the probability rate functions for the *dtts* that are in every state in a cycle is to traverse the cycle, applying equation 6-3, until the computation converges.

7. Evaluation

Although the theory points out the expected error for modeling a *dtt* probability rate function is the variance of $P_{true}(dtt, t, s_k)$, which is domain dependent, we would like to see how well our heuristic works in practice. We have generated a set of state diagrams with various temporal transitions. We find the state probabilities of the states using a stochastic method, a random walk in the state diagram. Then, we compare the results of the simulations with the state probabilities computed by the set of equations described earlier. As the probabilities calculated by the two methods are reasonably close, we are confident that the theory provides a reasonable framework to model *dtt* probability rate functions.

Over the 20+ experiments we ran, each for 50000 to 100000 trials, the errors ranged from about 1% to 8%. We have noticed that the errors tend to be larger for periodic functions (in which the probabilities rise and fall periodically). Qualitatively speaking, since we are essentially taking an average of all the "delayed" probability rate functions, the error between the predicted value and the "observed" value is therefore very sensitive to the shape of the "true" probability rate function. Details of our experiments are in the full paper.

We have also evaluated the backward compatibility of our new model to the non-probabilistic CIRCA originally developed by Musliner (Musliner, 1993). Space limitations preclude a full description here (see the full paper for more) but by describing temporal (and event) transitions as having a step-shaped probability rate function reaching a value strictly between 0 and 1, and a (guaranteed) action transition of having a step that reaches probability 1, our model generates the same “reachable state” space where reachable states have probability greater than zero.

8. Conclusions and Future Work

A framework for modeling the probabilistic nature of external events as functions of time has been developed. In contrast with traditional AI planners which concentrate on “what to do”, CIRCA-II emphasizes also on “when to do it” such that the probability of system failure is below a threshold. Moreover, we also present an algorithm to order the states by their likelihood so that the system can cut off the unlikely events from consideration in case of insufficient resources as illustrated in the previous section.

CIRCA-II is also a planner such that each state is only represented by one node in the state diagram regardless of the multiple paths leading to the state. Unlike a traditional AI planner, CIRCA-II either ignores or aggregates the information about how it reaches a state; this has significant benefits in devising a real-time control schedule, but also is a weakness. To plan only a single action for a group of states, CIRCA-II is considering only the worst-case temporal transitions (or, in the case of dependent temporal transitions, the worst-case temporal transition chains). There is a class of problems that CIRCA-II cannot find the solutions for because of its overly conservative philosophy. So, CIRCA-II is sound but not complete. There is ongoing research to address this dilemma.

9. Acknowledgements

The authors gratefully acknowledge the detailed comments and directions suggested by their Honeywell colleagues, Robert Goldman and David Musliner.

10. References

E. M. Atkins, E. H. Durfee, and K. G. Shin, “Plan Development using Local Probabilistic Models,” in *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, August 1996.

E. M. Atkins, E. H. Durfee, K. G. Shin, “Detecting and Reacting to Unplanned-for World States,” *Proceedings of AAAI*, pp. 571-576, July 1997.

E. M. Atkins, "Plan Generation and Hard Real-Time Execution with Application to Safe, Autonomous Flight", *Ph.D. Thesis*, *The University Of Michigan,* 1999.

C. Boutilier, T. Dean, and S. Hanks, “Planning Under Uncertainty: Structural Assumptions and Computational Leverage,” *EWSP*, 1995

T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, “Planning with Deadlines in Stochastic Domains,” *Proceedings of AAAI,* pp. 574-579, July 1993.

J. G. Kemeny, and J. L. Snell, “*Finite Markov Chanis,*” p.52-52, 1960.

N. K. Kushmerick, S. Hanks, D. Weld, “An Algorithm for Probabilistic Least-Commitment Planning,” *Proceedings Of AAAI,* pp. 1073-1078, July 1994.

M. L. Littman, T. L. Dean, and L. P. Kaelbling, “On the Complexity of Solving Markov Decision Problems,” *Proceedings of UAI-95,* August 1995.

G. F. Lawler, “Introduction to Stochastic Processes,” pp. 55-57, 1995.

D. J. Musliner, “CIRCA: The Cooperative Intelligent Real-Time Control Architecture,” *Ph.D. Thesis*, *The University Of Michigan,* 1993.

D. J. Musliner, E. H. Durfee, and K. G. Shin, “World Modeling for the Dynamic Construction of Real-Time Control Plans,” *Artificial Intelligence*, vol. 74, pp. 83-127, 1995.